Effiziente Algorithmen (SS2015)

Kapitel 7
Approximation II

Walter Unger

Lehrstuhl für Informatik 1

16:41 Uhr, den 29. November 2018

Inhalt I

Definition

Definition (Set-Cover-Problem)

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i \in \{1,2,\ldots,m\}} S_i = X$.

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i\in\{1,2,\ldots,m\}} S_i = X$.
- Für jede Menge S_i einen Kostenwert $c_i \in \mathbb{Q}$.

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i \in \{1,2,\ldots,m\}} S_i = X$.
- Für jede Menge S_i einen Kostenwert $c_i \in \mathbb{Q}$.
- Gesucht ist:

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i \in \{1,2,\ldots,m\}} S_i = X$.
- Für jede Menge S_i einen Kostenwert $c_i \in \mathbb{Q}$.
- Gesucht ist:
  - $A \subseteq \{1, 2, \ldots, m\}$ mit

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i \in \{1,2,\ldots,m\}} S_i = X$.
- Für jede Menge S_i einen Kostenwert $c_i \in \mathbb{Q}$.
- Gesucht ist:
  - $A \subseteq \{1, 2, \ldots, m\}$ mit
  - $\cup_{i \in A} S_i = X$ und

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i \in \{1,2,\ldots,m\}} S_i = X$.
- Für jede Menge S_i einen Kostenwert $c_i \in \mathbb{Q}$.
- Gesucht ist:
  - $A \subseteq \{1, 2, \ldots, m\}$ mit
  - $\cup_{i \in A} S_i = X$ und
  - $\text{cost}(A) = \sum_{i \in A} c_i$ ist minimal.

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i\in\{1,2,\ldots,m\}} S_i = X$.
- Für jede Menge S_i einen Kostenwert $c_i \in \mathbb{Q}$.
- Gesucht ist:
  - $A \subseteq \{1, 2, \ldots, m\}$ mit
  - $\cup_{i\in A} S_i = X$ und
  - $\text{cost}(A) = \sum_{i\in A} c_i$ ist minimal.

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i \in \{1,2,\ldots,m\}} S_i = X$.
- Für jede Menge S_i einen Kostenwert $c_i \in \mathbb{Q}$.
- Gesucht ist:
  - $A \subseteq \{1, 2, \ldots, m\}$ mit
  - $\cup_{i \in A} S_i = X$ und
  - $\text{cost}(A) = \sum_{i \in A} c_i$ ist minimal.

- Grundmenge soll also kostengünstig überdeckt werden.

Definition

Definition (Set-Cover-Problem)

- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i \in \{1,2,\ldots,m\}} S_i = X$.
- Für jede Menge S_i einen Kostenwert $c_i \in \mathbb{Q}$.
- Gesucht ist:
  - $A \subseteq \{1, 2, \ldots, m\}$ mit
  - $\cup_{i \in A} S_i = X$ und
  - $\text{cost}(A) = \sum_{i \in A} c_i$ ist minimal.

- Grundmenge soll also kostengünstig überdeckt werden.
- Beispiel: Kostengünstiges Arbeitsteam, welche alle nötigen Fähigkeiten hat.

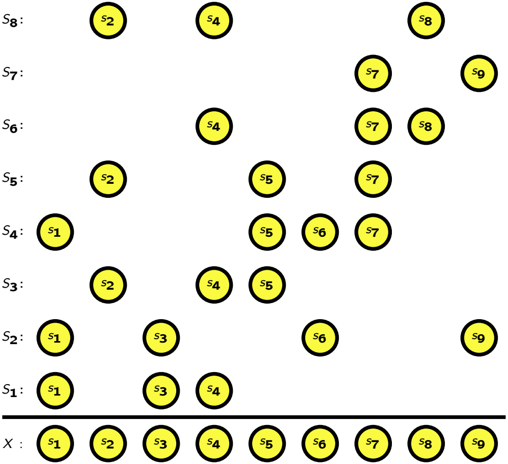
Definition

Definition (Set-Cover-Problem)

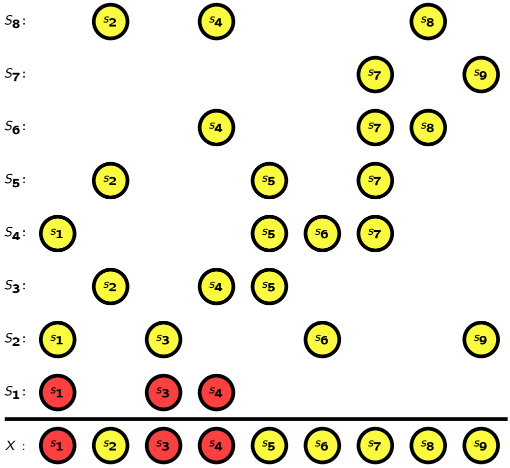
- Grundmenge X mit n Elementen.
- m Teilmengen $S_1, S_2, \ldots, S_m$ mit $\cup_{i \in \{1,2,\ldots,m\}} S_i = X$.
- Für jede Menge S_i einen Kostenwert $c_i \in \mathbb{Q}$.
- Gesucht ist:
  - $A \subseteq \{1, 2, \ldots, m\}$ mit
  - $\cup_{i \in A} S_i = X$ und
  - $\text{cost}(A) = \sum_{i \in A} c_i$ ist minimal.

- Grundmenge soll also kostengünstig überdeckt werden.
- Beispiel: Kostengünstiges Arbeitsteam, welche alle nötigen Fähigkeiten hat.
- Entspricht dem Vertex-Cover-Problem auf Hypergraphen (Hitting-Set-Problem).

Beispiel


S_8: s_2 s_4 s_8
S_7: s_7 s_9
S_6: s_4 s_7 s_8
S_5: s_2 s_5 s_7
S_4: s_1 s_5 s_6 s_7
S_3: s_2 s_4 s_5
S_2: s_1 s_3 s_6 s_9
S_1: s_1 s_3 s_4
X : s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9

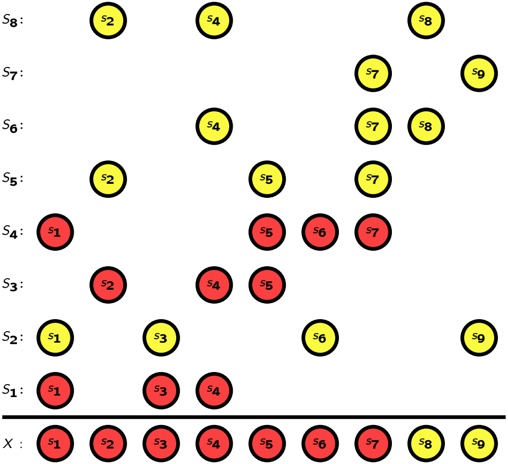
Beispiel


S_8: s_2 s_4 s_8
S_7: s_7 s_9
S_6: s_4 s_7 s_8
S_5: s_2 s_5 s_7
S_4: s_1 s_5 s_6 s_7
S_3: s_2 s_4 s_5
S_2: s_1 s_3 s_6 s_9
S_1: s_1 s_3 s_4
X : s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9

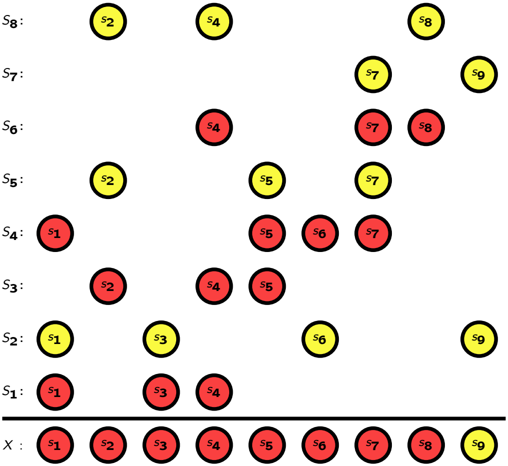
Beispiel

S_8:		s_2		s_4				s_8	
S_7:							s_7		s_9
S_6:				s_4			s_7	s_8	
S_5:		s_2			s_5		s_7		
S_4:	s_1				s_5	s_6	s_7		
S_3:		s_2		s_4	s_5				
S_2:	s_1		s_3			s_6			s_9
S_1:	s_1		s_3	s_4					
X :	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9

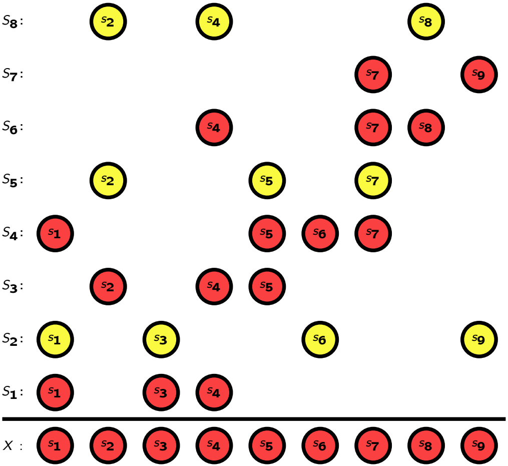
Beispiel


S_8: s_2 s_4 s_8
S_7: s_7 s_9
S_6: s_4 s_7 s_8
S_5: s_2 s_5 s_7
S_4: s_1 s_5 s_6 s_7
S_3: s_2 s_4 s_5
S_2: s_1 s_3 s_6 s_9
S_1: s_1 s_3 s_4
X : s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9

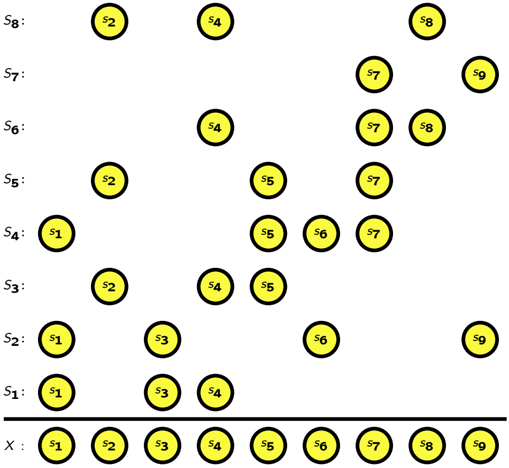
Beispiel


S_8: s_2 s_4 s_8
S_7: s_7 s_9
S_6: s_4 s_7 s_8
S_5: s_2 s_5 s_7
S_4: s_1 s_5 s_6 s_7
S_3: s_2 s_4 s_5
S_2: s_1 s_3 s_6 s_9
S_1: s_1 s_3 s_4
X : s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9

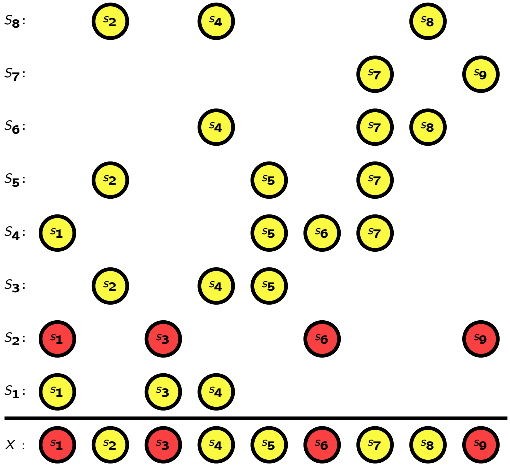
Beispiel (Kosten: 5)



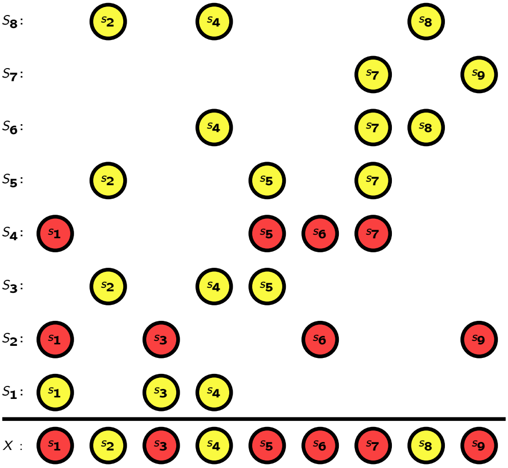
Beispiel nochmal



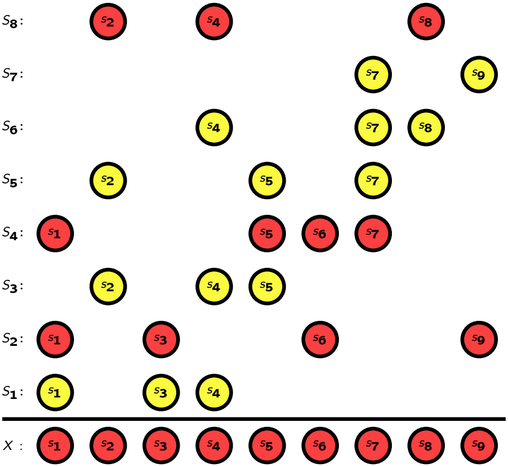
Beispiel nochmal



Beispiel nochmal



Beispiel nochmal (Kosten: 3)



Komplexität

Theorem

Die Entscheidungsvariante vom Set-Cover-Problem ist in $\mathcal{NPC}$.

Beweis: Einfache Reduktion vom Vertex-Cover-Problem:

Komplexität

Theorem

Die Entscheidungsvariante vom Set-Cover-Problem ist in $\mathcal{NPC}$.

Beweis: Einfache Reduktion vom Vertex-Cover-Problem:

Komplexität

Theorem

Die Entscheidungsvariante vom Set-Cover-Problem ist in $\mathcal{NPC}$.

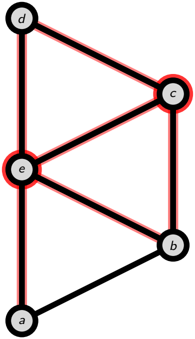
Beweis: Einfache Reduktion vom Vertex-Cover-Problem:

Komplexität

Theorem

Die Entscheidungsvariante vom Set-Cover-Problem ist in $\mathcal{NPC}$.

Beweis: Einfache Reduktion vom Vertex-Cover-Problem:

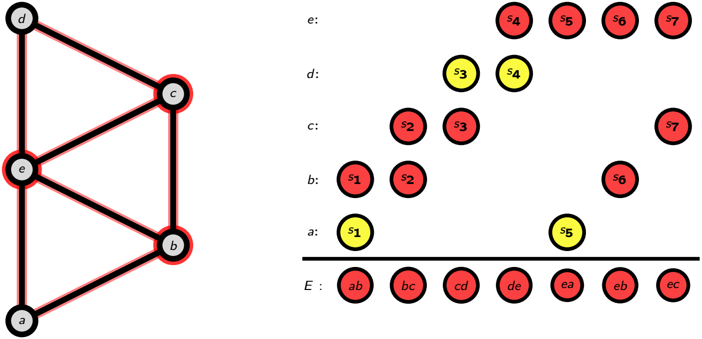


Komplexität

Theorem

Die Entscheidungsvariante vom Set-Cover-Problem ist in $\mathcal{NPC}$.

Beweis: Einfache Reduktion vom Vertex-Cover-Problem:



Aufbau der Idee

- Das Set-Cover-Problem hat wenig Struktur, die man nutzen könnte.

Aufbau der Idee

- Das Set-Cover-Problem hat wenig Struktur, die man nutzen könnte.
- Einzige Idee: versuche viele Elemente kostengünstig abzudecken.

Aufbau der Idee

- Das Set-Cover-Problem hat wenig Struktur, die man nutzen könnte.
- Einzige Idee: versuche viele Elemente kostengünstig abzudecken.
- Oder: versuche die Kosten pro Element klein zu halten.

Aufbau der Idee

- Das Set-Cover-Problem hat wenig Struktur, die man nutzen könnte.
- Einzige Idee: versuche viele Elemente kostengünstig abzudecken.
- Oder: versuche die Kosten pro Element klein zu halten.
- Also werden wir einen Greedy-Algorithmus entwickeln.

Aufbau der Idee

- Das Set-Cover-Problem hat wenig Struktur, die man nutzen könnte.
- Einzige Idee: versuche viele Elemente kostengünstig abzudecken.
- Oder: versuche die Kosten pro Element klein zu halten.
- Also werden wir einen Greedy-Algorithmus entwickeln.
- Auswahl der Menge S_i über:

$$\frac{\text{Kosten von } S_i}{\text{Anzahl der von } S_i \text{ neu überdeckten Elemente}}$$

Aufbau der Idee

- Das Set-Cover-Problem hat wenig Struktur, die man nutzen könnte.
- Einzige Idee: versuche viele Elemente kostengünstig abzudecken.
- Oder: versuche die Kosten pro Element klein zu halten.
- Also werden wir einen Greedy-Algorithmus entwickeln.
- Auswahl der Menge S_i über:

$$\frac{\text{Kosten von } S_i}{\text{Anzahl der von } S_i \text{ neu überdeckten Elemente}}$$

- Wähle Menge S_i, wo obiger Ausdrucke minimal ist.

Greedy-Algorithmus

1. Eingabe $X, S_1, S_2, \ldots S_m$.

Greedy-Algorithmus

1. Eingabe $X, S_1, S_2, \ldots S_m$.
2. Setze $A = \emptyset$.

Greedy-Algorithmus

1. Eingabe $X, S_1, S_2, \ldots S_m$.
2. Setze $A = \emptyset$.
3. Solange $\cup_{j \in A} S_j \neq X$ wiederhole:

Greedy-Algorithmus

1. Eingabe $X, S_1, S_2, \ldots S_m$.
2. Setze $A = \emptyset$.
3. Solange $\cup_{j \in A} S_j \neq X$ wiederhole:
   1. Bestimme für alle $i \in \{1, 2, \cdots, m\} \setminus A$:

$$r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}.$$

Greedy-Algorithmus

1. Eingabe $X, S_1, S_2, \ldots S_m$.
2. Setze $A = \emptyset$.
3. Solange $\cup_{j \in A} S_j \neq X$ wiederhole:
   1. Bestimme für alle $i \in \{1, 2, \cdots, m\} \setminus A$:
   $$r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}.$$
   2. Wähle i mit $r_A(i)$ minimal.

Greedy-Algorithmus

1. Eingabe $X, S_1, S_2, \ldots S_m$.
2. Setze $A = \emptyset$.
3. Solange $\cup_{j \in A} S_j \neq X$ wiederhole:
   1. Bestimme für alle $i \in \{1, 2, \cdots, m\} \setminus A$:
   $$r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}.$$
   2. Wähle i mit $r_A(i)$ minimal.
   3. Setze $A = A \cup \{i\}$.

Analyse

- Verteile die Kosten bei der Wahl einer Menge S_i auf die Elemente, die durch S_i neu überdeckt werden.

Analyse

- Verteile die Kosten bei der Wahl einer Menge S_i auf die Elemente, die durch S_i neu überdeckt werden.
- D.h. in jeder Schleife erhält jedes dieser Elemente den folgenden Wert zugeordnet:

$$\frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|} = r_A(i)$$

Analyse

- Verteile die Kosten bei der Wahl einer Menge S_i auf die Elemente, die durch S_i neu überdeckt werden.
- D.h. in jeder Schleife erhält jedes dieser Elemente den folgenden Wert zugeordnet:
$$\frac{c_i}{|S_i \setminus \cup_{j\in A} S_j|} = r_A(i)$$
- Nun betrachten wir <u>einzeln</u> die hinzugefügten Elemente.

Analyse

- Verteile die Kosten bei der Wahl einer Menge S_i auf die Elemente, die durch S_i neu überdeckt werden.
- D.h. in jeder Schleife erhält jedes dieser Elemente den folgenden Wert zugeordnet:
$$\frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|} = r_A(i)$$
- Nun betrachten wir einzeln die hinzugefügten Elemente.
- Sei also für $k \in \{1, 2, \cdots, n\}$ x_k das k-te Element.

Analyse

- Verteile die Kosten bei der Wahl einer Menge S_i auf die Elemente, die durch S_i neu überdeckt werden.
- D.h. in jeder Schleife erhält jedes dieser Elemente den folgenden Wert zugeordnet:

$$\frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|} = r_A(i)$$

- Nun betrachten wir einzeln die hinzugefügten Elemente.
- Sei also für $k \in \{1, 2, \cdots, n\}$ x_k das k-te Element.
- Seien weiter $c(x_k)$ die relativen Kosten, die x_k zugeordnet worden sind.

Analyse

- Verteile die Kosten bei der Wahl einer Menge S_i auf die Elemente, die durch S_i neu überdeckt werden.
- D.h. in jeder Schleife erhält jedes dieser Elemente den folgenden Wert zugeordnet:
$$\frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|} = r_A(i)$$
- Nun betrachten wir einzeln die hinzugefügten Elemente.
- Sei also für $k \in \{1, 2, \cdots, n\}$ x_k das k-te Element.
- Seien weiter $c(x_k)$ die relativen Kosten, die x_k zugeordnet worden sind.

Analyse

- Verteile die Kosten bei der Wahl einer Menge S_i auf die Elemente, die durch S_i neu überdeckt werden.
- D.h. in jeder Schleife erhält jedes dieser Elemente den folgenden Wert zugeordnet:

$$\frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|} = r_A(i)$$

- Nun betrachten wir einzeln die hinzugefügten Elemente.
- Sei also für $k \in \{1, 2, \cdots, n\}$ x_k das k-te Element.
- Seien weiter $c(x_k)$ die relativen Kosten, die x_k zugeordnet worden sind.

Lemma

Für $k \in \{1, 2, \cdots, n\}$ gilt $c(x_k) \leqslant opt/(n - k + 1)$, wobei opt die Kosten eines optimalen Set-Covers sind.

Beweis

$c(x_k) \leqslant opt/(n-k+\mathbf{1})$

- Sei A die Auswahl, die vorher gewählt wurde.

Beweis

$c(x_k) \leqslant opt/(n - k + \mathbf{1})$

- Sei A die Auswahl, die vorher gewählt wurde.
- Sei weiter $i \in \{1, 2, \cdots, n\}$ der Index der Menge S_i, die x_k erstmalig abdeckt.

Beweis

$c(x_k) \leqslant opt/(n-k+1)$

- Sei A die Auswahl, die vorher gewählt wurde.
- Sei weiter $i \in \{1, 2, \cdots, n\}$ der Index der Menge S_i, die x_k erstmalig abdeckt.
- Nun schätzen wir *opt* ab:

Beweis

$c(x_k) \leqslant opt/(n-k+1)$

- Sei A die Auswahl, die vorher gewählt wurde.
- Sei weiter $i \in \{1, 2, \cdots, n\}$ der Index der Menge S_i, die x_k erstmalig abdeckt.
- Nun schätzen wir *opt* ab:
  - Für $j \in \{1, 2, \cdots, n\} \setminus A$ gilt: $r_A(j) \geqslant r_A(i)$.

Beweis

$c(x_k) \leqslant opt/(n - k + \mathbf{1})$

- Sei A die Auswahl, die vorher gewählt wurde.
- Sei weiter $i \in \{1, 2, \cdots, n\}$ der Index der Menge S_i, die x_k erstmalig abdeckt.
- Nun schätzen wir *opt* ab:
  - Für $j \in \{1, 2, \cdots, n\} \setminus A$ gilt: $r_A(j) \geqslant r_A(i)$.
  - Setze: $X' = X \setminus \cup_{j \in A} S_j$, d.h. X' ist noch nicht abgedeckt.

Beweis

$c(x_k) \leqslant opt/(n-k+\mathbf{1})$

- Sei A die Auswahl, die vorher gewählt wurde.
- Sei weiter $i \in \{1, 2, \cdots, n\}$ der Index der Menge S_i, die x_k erstmalig abdeckt.
- Nun schätzen wir *opt* ab:
  - Für $j \in \{1, 2, \cdots, n\} \setminus A$ gilt: $r_A(j) \geqslant r_A(i)$.
  - Setze: $X' = X \setminus \cup_{j \in A} S_j$, d.h. X' ist noch nicht abgedeckt.
  - Kein Element $j \in X'$ kann mit relativen Kosten kleiner als $r_A(i)$ abgedeckt werden.

Beweis

$c(x_k) \leqslant opt/(n-k+1)$

- Sei A die Auswahl, die vorher gewählt wurde.
- Sei weiter $i \in \{1, 2, \cdots, n\}$ der Index der Menge S_i, die x_k erstmalig abdeckt.
- Nun schätzen wir *opt* ab:
  - Für $j \in \{1, 2, \cdots, n\} \setminus A$ gilt: $r_A(j) \geqslant r_A(i)$.
  - Setze: $X' = X \setminus \cup_{j \in A} S_j$, d.h. X' ist noch nicht abgedeckt.
  - Kein Element $j \in X'$ kann mit relativen Kosten kleiner als $r_A(i)$ abgedeckt werden.
  - Damit ist die Summe der relativen Kosten von X' mindestens: $(n-k+1) \cdot r_A(i)$.

Beweis

$c(x_k) \leqslant opt/(n-k+1)$

- Sei A die Auswahl, die vorher gewählt wurde.
- Sei weiter $i \in \{1, 2, \cdots, n\}$ der Index der Menge S_i, die x_k erstmalig abdeckt.
- Nun schätzen wir *opt* ab:
  - Für $j \in \{1, 2, \cdots, n\} \setminus A$ gilt: $r_A(j) \geqslant r_A(i)$.
  - Setze: $X' = X \setminus \cup_{j \in A} S_j$, d.h. X' ist noch nicht abgedeckt.
  - Kein Element $j \in X'$ kann mit relativen Kosten kleiner als $r_A(i)$ abgedeckt werden.
  - Damit ist die Summe der relativen Kosten von X' mindestens: $(n-k+1) \cdot r_A(i)$.
  - Jede mögliche Auswahl, die X' abdeckt, hat damit Kosten von mindestens: $(n-k+1) \cdot r_A(i)$.

Beweis

- Sei A die Auswahl, die vorher gewählt wurde.
- Sei weiter $i \in \{1, 2, \cdots, n\}$ der Index der Menge S_i, die x_k erstmalig abdeckt.
- Nun schätzen wir *opt* ab:
  - Für $j \in \{1, 2, \cdots, n\} \setminus A$ gilt: $r_A(j) \geqslant r_A(i)$.
  - Setze: $X' = X \setminus \cup_{j \in A} S_j$, d.h. X' ist noch nicht abgedeckt.
  - Kein Element $j \in X'$ kann mit relativen Kosten kleiner als $r_A(i)$ abgedeckt werden.
  - Damit ist die Summe der relativen Kosten von X' mindestens: $(n-k+1) \cdot r_A(i)$.
  - Jede mögliche Auswahl, die X' abdeckt, hat damit Kosten von mindestens: $(n-k+1) \cdot r_A(i)$.
- Damit gilt: $opt \geqslant (n-k+1) \cdot r_A(i) = (n-k+1) \cdot c(x_k)$ und

$$c(x_k) \leqslant opt/(n-k+1).$$

Güte der Approximation

$c(x_k) \leqslant opt/(n-k+\mathbf{1})$

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

Güte der Approximation

$c(x_k) \leqslant opt/(n - k + \mathbf{1})$

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

- H_n ist die n-te Harmonische Zahl.

Güte der Approximation

$c(x_k) \leqslant opt/(n-k+1)$

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

- H_n ist die n-te Harmonische Zahl.
- $H_n = \sum_{i=1}^{n} \frac{1}{i}$.

Güte der Approximation

$c(x_k) \leqslant opt/(n - k + 1)$

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

- H_n ist die n-te Harmonische Zahl.
- $H_n = \sum_{i=1}^{n} \frac{1}{i}$.
- Es gilt: $\log(n+1) \leqslant H_n \leqslant \log n + 1$.

Güte der Approximation

$c(x_k) \leqslant opt/(n - k + \mathbf{1})$

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

- H_n ist die n-te Harmonische Zahl.
- $H_n = \sum_{i=1}^{n} \frac{1}{i}$.
- Es gilt: $\log(n+1) \leqslant H_n \leqslant \log n + 1$.
- Beweis des Theorems:

Güte der Approximation

$c(x_k) \leqslant opt/(n - k + \mathbf{1})$

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

- H_n ist die n-te Harmonische Zahl.
- $H_n = \sum_{i=1}^{n} \frac{1}{i}$.
- Es gilt: $\log(n+1) \leqslant H_n \leqslant \log n + 1$.
- Beweis des Theorems:
  - Bilde Summe über alle Elemente.

Güte der Approximation

$c(x_k) \leqslant opt/(n-k+1)$

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

- H_n ist die n-te Harmonische Zahl.
- $H_n = \sum_{i=1}^{n} \frac{1}{i}$.
- Es gilt: $\log(n+1) \leqslant H_n \leqslant \log n + 1$.
- Beweis des Theorems:
  - Bilde Summe über alle Elemente.
  - Nutze obiges Lemma:

$$\sum_{i=1}^{n} \frac{opt}{n-i+1} = \sum_{i=1}^{n} \frac{opt}{i} = opt \cdot H_n.$$

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
    - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.
- Nun sind noch die Kosten zu wählen:

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n+1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \backslash \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \backslash \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1}{n}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \backslash \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \backslash \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) < \frac{1}{n}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) < \frac{1}{n}$.
- Damit gilt: $r_{\{1\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{2,3,\ldots,n\}|} = \frac{1}{n-1}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) < \frac{1}{n}$.
- Damit gilt: $r_{\{1\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{2,3,\ldots,n\}|} = \frac{1}{n-1}$.
- Damit der Algorithmus S_2 wählt, muss gelten: $r_{\{1\}}(2) < \frac{1}{n-1}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n+1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) < \frac{1}{n}$.
- Damit gilt: $r_{\{1\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{2,3,\ldots,n\}|} = \frac{1}{n-1}$.
- Damit der Algorithmus S_2 wählt, muss gelten: $r_{\{1\}}(2) < \frac{1}{n-1}$.
- Damit gilt: $r_{\{1,2\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{3,4,\ldots,n\}|} = \frac{1}{n-2}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \backslash \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \backslash \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) < \frac{1}{n}$.
- Damit gilt: $r_{\{1\}}(m) = \frac{1}{|S_i \backslash \cup_{i \in A} S_i|} = \frac{1}{|\{2,3,\ldots,n\}|} = \frac{1}{n-1}$.
- Damit der Algorithmus S_2 wählt, muss gelten: $r_{\{1\}}(2) < \frac{1}{n-1}$.
- Damit gilt: $r_{\{1,2\}}(m) = \frac{1}{|S_i \backslash \cup_{i \in A} S_i|} = \frac{1}{|\{3,4,\ldots,n\}|} = \frac{1}{n-2}$.
- Damit der Algorithmus S_3 wählt, muss gelten: $r_{\{1,2\}}(3) < \frac{1}{n-2}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \dots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \dots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n+1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\dots,n\}|} = \frac{1}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) < \frac{1}{n}$.
- Damit gilt: $r_{\{1\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{2,3,\dots,n\}|} = \frac{1}{n-1}$.
- Damit der Algorithmus S_2 wählt, muss gelten: $r_{\{1\}}(2) < \frac{1}{n-1}$.
- Damit gilt: $r_{\{1,2\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{3,4,\dots,n\}|} = \frac{1}{n-2}$.
- Damit der Algorithmus S_3 wählt, muss gelten: $r_{\{1,2\}}(3) < \frac{1}{n-2}$.
- Zur Vereinfachung wählen wir im Folgenden: $c_m = 1{+}\varepsilon$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n+1)$.
- Nun sind noch die Kosten zu wählen:

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n+1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1+\varepsilon$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n+1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1+\varepsilon$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1+\varepsilon}{n}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n+1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1+\varepsilon$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1+\varepsilon}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) \leqslant \frac{1}{n}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1+\varepsilon$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1+\varepsilon}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) \leqslant \frac{1}{n}$.
- Damit gilt: $r_{\{1\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{2,3,\ldots,n\}|} = \frac{1+\varepsilon}{n-1}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \backslash \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n+1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1{+}\varepsilon$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \backslash \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1+\varepsilon}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) \leqslant \frac{1}{n}$.
- Damit gilt: $r_{\{1\}}(m) = \frac{1}{|S_i \backslash \cup_{i \in A} S_i|} = \frac{1}{|\{2,3,\ldots,n\}|} = \frac{1+\varepsilon}{n-1}$.
- Damit der Algorithmus S_2 wählt, muss gelten: $r_{\{1\}}(2) \leqslant \frac{1}{n-1}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n+1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1+\varepsilon$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1+\varepsilon}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) \leqslant \frac{1}{n}$.
- Damit gilt: $r_{\{1\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{2,3,\ldots,n\}|} = \frac{1+\varepsilon}{n-1}$.
- Damit der Algorithmus S_2 wählt, muss gelten: $r_{\{1\}}(2) \leqslant \frac{1}{n-1}$.
- Damit gilt: $r_{\{1,2\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{3,4,\ldots,n\}|} = \frac{1+\varepsilon}{n-2}$.

Frage: wie gut ist die Abschätzung

- Suche ein schlechtes Beispiel für obigen Algorithmus.
- Erinnerung: $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{j \in A} S_j|}$.
- Fehler soll möglichst groß sein:
  - Greedy Lösung: $A = \{1, 2, \ldots, m-1\}$
  - Optimale Lösung $opt = \{m\}$
  - Daher $S_m = \{1, 2, \ldots, n\}$ und $S_i = \{i\}$ für $1 \leqslant i \leqslant n$ $(m = n + 1)$.
- Nun sind noch die Kosten zu wählen:
- Setze $c_m = 1+\varepsilon$.
- Damit gilt: $r_\emptyset(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{1,2,\ldots,n\}|} = \frac{1+\varepsilon}{n}$.
- Damit der Algorithmus S_1 wählt, muss gelten: $r_\emptyset(1) \leqslant \frac{1}{n}$.
- Damit gilt: $r_{\{1\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{2,3,\ldots,n\}|} = \frac{1+\varepsilon}{n-1}$.
- Damit der Algorithmus S_2 wählt, muss gelten: $r_{\{1\}}(2) \leqslant \frac{1}{n-1}$.
- Damit gilt: $r_{\{1,2\}}(m) = \frac{1}{|S_i \setminus \cup_{i \in A} S_i|} = \frac{1}{|\{3,4,\ldots,n\}|} = \frac{1+\varepsilon}{n-2}$.
- Damit der Algorithmus S_3 wählt, muss gelten: $r_{\{1,2\}}(3) \leqslant \frac{1}{n-2}$.

Beispiel zur Abschätzung der Güte

- Damit erhalten wir das folgende schlechte Beispiel für obigen Algorithmus:

Beispiel zur Abschätzung der Güte

- Damit erhalten wir das folgende schlechte Beispiel für obigen Algorithmus:
  - $X = \{1, 2, \ldots, n\}$

Beispiel zur Abschätzung der Güte

- Damit erhalten wir das folgende schlechte Beispiel für obigen Algorithmus:
  - $X = \{1, 2, \ldots, n\}$
  - $S_m = \{1, 2, \ldots, n\}$ mit $m = n + 1$.

Beispiel zur Abschätzung der Güte

- Damit erhalten wir das folgende schlechte Beispiel für obigen Algorithmus:
  - $X = \{1, 2, \ldots, n\}$
  - $S_m = \{1, 2, \ldots, n\}$ mit $m = n + 1$.
  - $S_i = \{i\}$ für $1 \leqslant i \leqslant n$.

Beispiel zur Abschätzung der Güte

- Damit erhalten wir das folgende schlechte Beispiel für obigen Algorithmus:
  - $X = \{1, 2, \ldots, n\}$
  - $S_m = \{1, 2, \ldots, n\}$ mit $m = n + 1$.
  - $S_i = \{i\}$ für $1 \leqslant i \leqslant n$.
  - Setze $c_m = 1 + \varepsilon$.

Beispiel zur Abschätzung der Güte

- Damit erhalten wir das folgende schlechte Beispiel für obigen Algorithmus:
  - $X = \{1, 2, \ldots, n\}$
  - $S_m = \{1, 2, \ldots, n\}$ mit $m = n + 1$.
  - $S_i = \{i\}$ für $1 \leqslant i \leqslant n$.
  - Setze $c_m = 1 + \varepsilon$.
  - Setze $c_i = \frac{1}{n-i+1}$.

Beispiel zur Abschätzung der Güte

- Damit erhalten wir das folgende schlechte Beispiel für obigen Algorithmus:
  - $X = \{1, 2, \ldots, n\}$
  - $S_m = \{1, 2, \ldots, n\}$ mit $m = n + 1$.
  - $S_i = \{i\}$ für $1 \leqslant i \leqslant n$.
  - Setze $c_m = 1 + \varepsilon$.
  - Setze $c_i = \frac{1}{n-i+1}$.
- Damit gilt für dieses Beispiel:

Beispiel zur Abschätzung der Güte

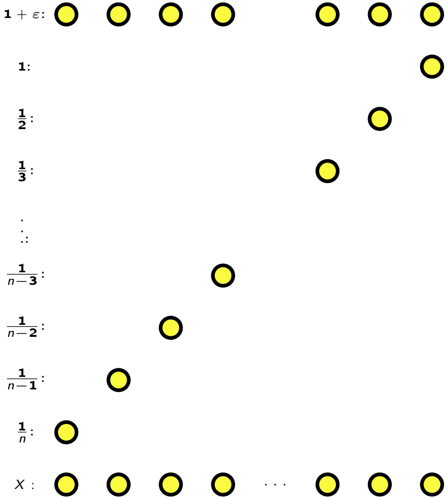
- Damit erhalten wir das folgende schlechte Beispiel für obigen Algorithmus:
  - $X = \{1, 2, \ldots, n\}$
  - $S_m = \{1, 2, \ldots, n\}$ mit $m = n + 1$.
  - $S_i = \{i\}$ für $1 \leqslant i \leqslant n$.
  - Setze $c_m = 1 + \varepsilon$.
  - Setze $c_i = \frac{1}{n-i+1}$.
- Damit gilt für dieses Beispiel:
  - Der Greedy Algorithmus wird nacheinander die Mengen $S_1, S_2, \ldots, S_n$ wählen.

Beispiel zur Abschätzung der Güte

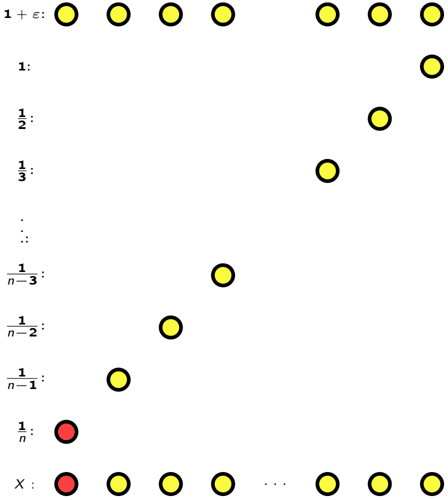
- Damit erhalten wir das folgende schlechte Beispiel für obigen Algorithmus:
  - $X = \{1, 2, \ldots, n\}$
  - $S_m = \{1, 2, \ldots, n\}$ mit $m = n + 1$.
  - $S_i = \{i\}$ für $1 \leqslant i \leqslant n$.
  - Setze $c_m = 1 + \varepsilon$.
  - Setze $c_i = \frac{1}{n-i+1}$.
- Damit gilt für dieses Beispiel:
  - Der Greedy Algorithmus wird nacheinander die Mengen $S_1, S_2, \ldots, S_n$ wählen.
  - Die optimale Lösung beinhaltet nur S_m.

Beispiel zur Güte

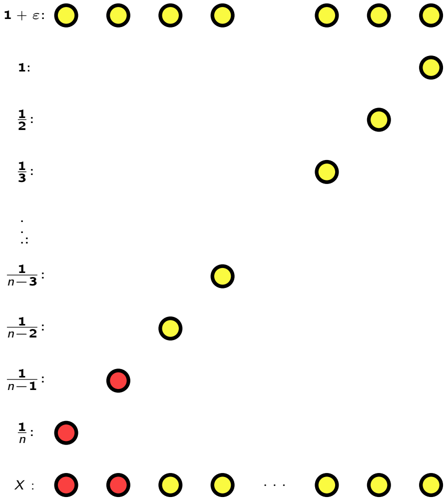
Beispiel zur Güte



Beispiel zur Güte

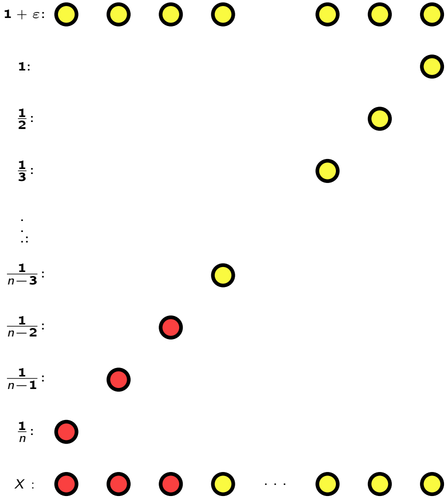


Beispiel zur Güte



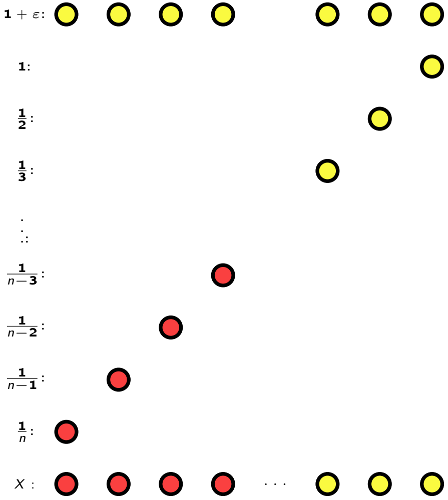
- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i \in A} S_i|}$.

Beispiel zur Güte



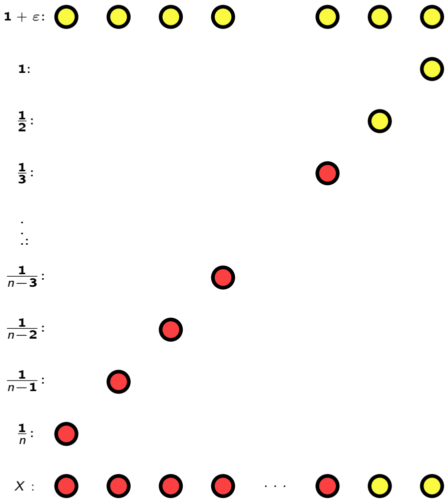
- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i \in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta$.

Beispiel zur Güte



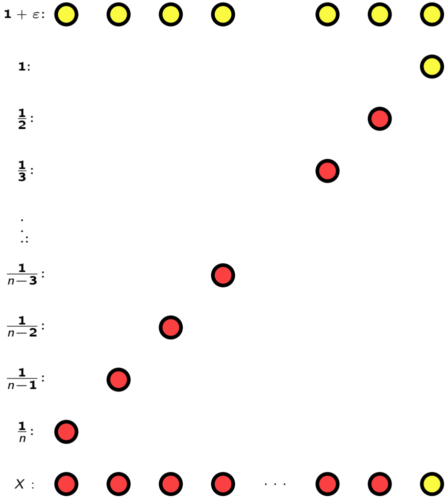
- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i \in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta.$
- $r_\emptyset(1) = \frac{1/n}{1} = \frac{1}{n}$

Beispiel zur Güte



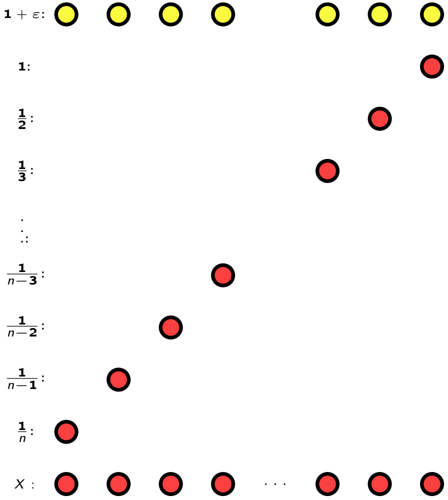
- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i\in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta$.
- $r_\emptyset(1) = \frac{1/n}{1} = \frac{1}{n}$
- $r_1(2) = \frac{1/(n-1)}{1} = \frac{1}{n-1}$

Beispiel zur Güte



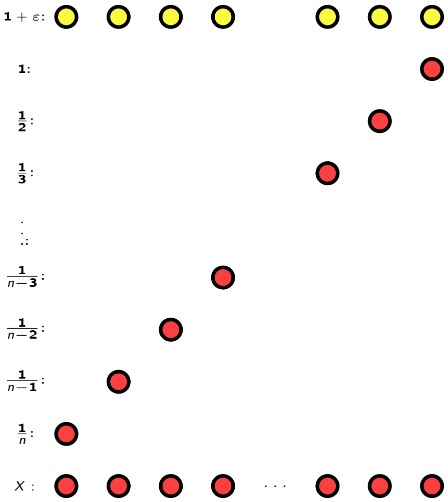
- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i \in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta$.
- $r_\emptyset(1) = \frac{1/n}{1} = \frac{1}{n}$
- $r_1(2) = \frac{1/(n-1)}{1} = \frac{1}{n-1}$
- $r_{1,2}(3) = \frac{1/(n-2)}{1} = \frac{1}{n-2}$

Beispiel zur Güte



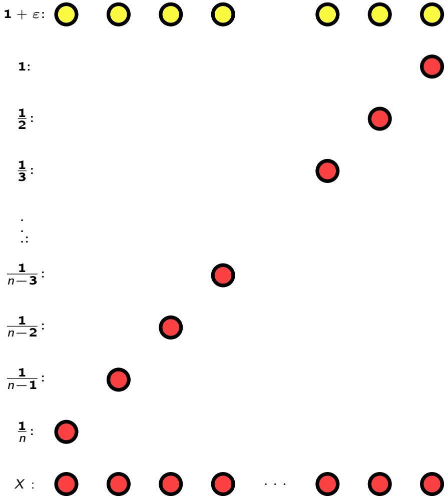
- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i \in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta$.
- $r_\emptyset(1) = \frac{1/n}{1} = \frac{1}{n}$
- $r_1(2) = \frac{1/(n-1)}{1} = \frac{1}{n-1}$
- $r_{1,2}(3) = \frac{1/(n-2)}{1} = \frac{1}{n-2}$
- $r_{1,2,3}(4) = \frac{1/(n-3)}{1} = \frac{1}{n-3}$

Beispiel zur Güte



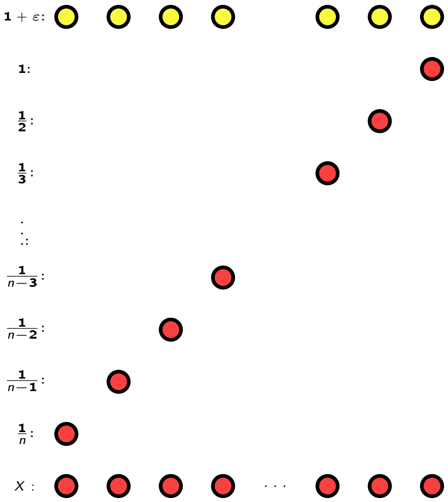
- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i \in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta.$
- $r_\emptyset(1) = \frac{1/n}{1} = \frac{1}{n}$
- $r_1(2) = \frac{1/(n-1)}{1} = \frac{1}{n-1}$
- $r_{1,2}(3) = \frac{1/(n-2)}{1} = \frac{1}{n-2}$
- $r_{1,2,3}(4) = \frac{1/(n-3)}{1} = \frac{1}{n-3}$
- $r_{1,2,\ldots,n-3}(n-2) = \frac{1/3}{1} = \frac{1}{3}$

Beispiel zur Güte



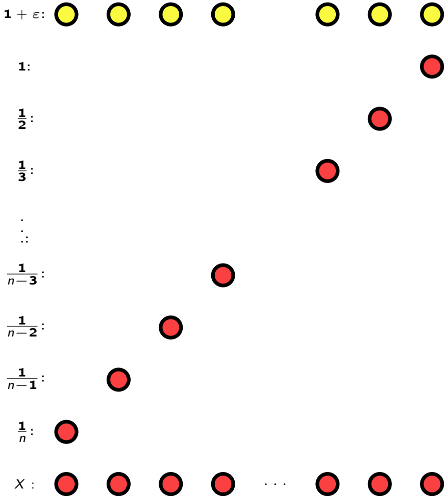
- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i\in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta$.
- $r_\emptyset(1) = \frac{1/n}{1} = \frac{1}{n}$
- $r_1(2) = \frac{1/(n-1)}{1} = \frac{1}{n-1}$
- $r_{1,2}(3) = \frac{1/(n-2)}{1} = \frac{1}{n-2}$
- $r_{1,2,3}(4) = \frac{1/(n-3)}{1} = \frac{1}{n-3}$
- $r_{1,2,\ldots,n-3}(n-2) = \frac{1/3}{1} = \frac{1}{3}$
- $r_{1,2,\ldots,n-2}(n-1) = \frac{1/2}{1} = \frac{1}{2}$

Beispiel zur Güte



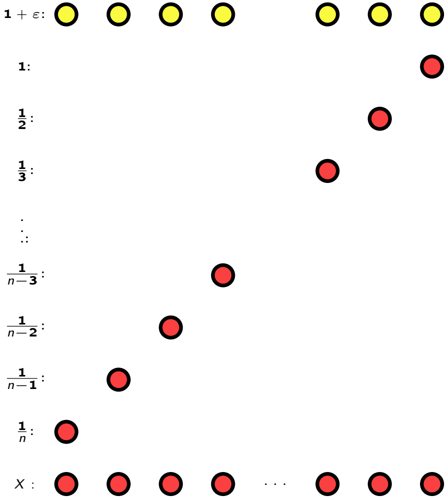
- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i \in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta$.
- $r_\emptyset(1) = \frac{1/n}{1} = \frac{1}{n}$
- $r_1(2) = \frac{1/(n-1)}{1} = \frac{1}{n-1}$
- $r_{1,2}(3) = \frac{1/(n-2)}{1} = \frac{1}{n-2}$
- $r_{1,2,3}(4) = \frac{1/(n-3)}{1} = \frac{1}{n-3}$
- $r_{1,2,\ldots,n-3}(n-2) = \frac{1/3}{1} = \frac{1}{3}$
- $r_{1,2,\ldots,n-2}(n-1) = \frac{1/2}{1} = \frac{1}{2}$
- $r_{1,2,\ldots,n-1}(n) = \frac{1}{1} = 1$

Beispiel zur Güte



- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i \in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta$.
- $r_\emptyset(1) = \frac{1/n}{1} = \frac{1}{n}$
- $r_1(2) = \frac{1/(n-1)}{1} = \frac{1}{n-1}$
- $r_{1,2}(3) = \frac{1/(n-2)}{1} = \frac{1}{n-2}$
- $r_{1,2,3}(4) = \frac{1/(n-3)}{1} = \frac{1}{n-3}$
- $r_{1,2,\ldots,n-3}(n-2) = \frac{1/3}{1} = \frac{1}{3}$
- $r_{1,2,\ldots,n-2}(n-1) = \frac{1/2}{1} = \frac{1}{2}$
- $r_{1,2,\ldots,n-1}(n) = \frac{1}{1} = 1$

Beispiel zur Güte



- $r_A(i) = \frac{c_i}{|S_i \setminus \cup_{i \in A} S_i|}$.
- $r_C(i+1) = \frac{1+\varepsilon}{n-|C|} = \frac{1}{n-|C|} + \delta.$
- $r_\emptyset(1) = \frac{1/n}{1} = \frac{1}{n}$
- $r_1(2) = \frac{1/(n-1)}{1} = \frac{1}{n-1}$
- $r_{1,2}(3) = \frac{1/(n-2)}{1} = \frac{1}{n-2}$
- $r_{1,2,3}(4) = \frac{1/(n-3)}{1} = \frac{1}{n-3}$
- $r_{1,2,\ldots,n-3}(n-2) = \frac{1/3}{1} = \frac{1}{3}$
- $r_{1,2,\ldots,n-2}(n-1) = \frac{1/2}{1} = \frac{1}{2}$
- $r_{1,2,\ldots,n-1}(n) = \frac{1}{1} = 1$

Güte der Approximation

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

Güte der Approximation

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

Güte der Approximation

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

Theorem

Es gibt eine Instanz, worauf der Greedy Algorithmus einen Approximationsfaktor von $(1-\varepsilon) \cdot H_n$ erreicht (für $\varepsilon > 0$).

Güte der Approximation

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

Theorem

Es gibt eine Instanz, worauf der Greedy Algorithmus einen Approximationsfaktor von $(1-\varepsilon)\cdot H_n$ erreicht (für $\varepsilon > 0$).

Güte der Approximation

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

Theorem

Es gibt eine Instanz, worauf der Greedy Algorithmus einen Approximationsfaktor von $(1-\varepsilon)\cdot H_n$ erreicht (für $\varepsilon > 0$).

Theorem (Feige 1995)

Es gibt keinen Algorithmus mit Approximationsfaktor $(1-\varepsilon)\cdot H_n$ für das Set-Cover-Problem, es sei denn $\mathcal{NP} = TIME(n^{O(\log\log n)})$.

Güte der Approximation

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

Theorem

Es gibt eine Instanz, worauf der Greedy Algorithmus einen Approximationsfaktor von $(1-\varepsilon)\cdot H_n$ erreicht (für $\varepsilon > 0$).

Theorem (Feige 1995)

Es gibt keinen Algorithmus mit Approximationsfaktor $(1-\varepsilon)\cdot H_n$ für das Set-Cover-Problem, es sei denn $\mathcal{NP} = TIME(n^{O(\log\log n)})$.

Güte der Approximation

Theorem

Der Greedy Algorithmus hat einen Approximationsfaktor von höchstens H_n.

Theorem

Es gibt eine Instanz, worauf der Greedy Algorithmus einen Approximationsfaktor von $(1-\varepsilon) \cdot H_n$ erreicht (für $\varepsilon > 0$).

Theorem (Feige 1995)

Es gibt keinen Algorithmus mit Approximationsfaktor $(1-\varepsilon) \cdot H_n$ für das Set-Cover-Problem, es sei denn $\mathcal{NP} = TIME(n^{O(\log\log n)})$.

Folgerung:

Damit ist dieser einfache Greedy Algorithmus der beste, der möglich ist.

Motivation

- Verteilung von Aufgaben

Motivation

- Verteilung von Aufgaben
- Aufgaben sind nicht teilbar (ansonsten wäre es zu einfach).

Motivation

- Verteilung von Aufgaben
- Aufgaben sind nicht teilbar (ansonsten wäre es zu einfach).
- Alle Aufgaben sollen möglichst schnell fertig werden.

Motivation

- Verteilung von Aufgaben
- Aufgaben sind nicht teilbar (ansonsten wäre es zu einfach).
- Alle Aufgaben sollen möglichst schnell fertig werden.
- Beispiel

Motivation

- Verteilung von Aufgaben
- Aufgaben sind nicht teilbar (ansonsten wäre es zu einfach).
- Alle Aufgaben sollen möglichst schnell fertig werden.
- Beispiel
  - Vor einer Party gibt es viele Arbeiten zu machen.

Motivation

- Verteilung von Aufgaben
- Aufgaben sind nicht teilbar (ansonsten wäre es zu einfach).
- Alle Aufgaben sollen möglichst schnell fertig werden.
- Beispiel
    - Vor einer Party gibt es viele Arbeiten zu machen.
    - Wenn jemand eine Arbeit hat, gibt er sie nicht mehr her.

Motivation

- Verteilung von Aufgaben
- Aufgaben sind nicht teilbar (ansonsten wäre es zu einfach).
- Alle Aufgaben sollen möglichst schnell fertig werden.
- Beispiel
  - Vor einer Party gibt es viele Arbeiten zu machen.
  - Wenn jemand eine Arbeit hat, gibt er sie nicht mehr her.
  - Erst wenn alle Arbeiten erledigt sind, fängt die Party an.

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $m \in \mathbb{N}$ (d.h. m identische Maschinen).

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $m \in \mathbb{N}$ (d.h. m identische Maschinen).
- Gesucht:

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $m \in \mathbb{N}$ (d.h. m identische Maschinen).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$,
    (d.h. Zuweisung der Jobs auf Maschinen).

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $m \in \mathbb{N}$ (d.h. m identische Maschinen).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$,
    (d.h. Zuweisung der Jobs auf Maschinen).
  - Mit $\max_{j \in \{1,2,\ldots,m\}} \sum_{i \in \{1,2,\ldots,n\} \wedge f(i)=j} p_i$ ist minimal.

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $m \in \mathbb{N}$ (d.h. m identische Maschinen).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$,
    (d.h. Zuweisung der Jobs auf Maschinen).
  - Mit $\max_{j \in \{1,2,\ldots,m\}} \sum_{i \in \{1,2,\ldots,n\} \wedge f(i)=j} p_i$ ist minimal.

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $m \in \mathbb{N}$ (d.h. m identische Maschinen).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$,
    (d.h. Zuweisung der Jobs auf Maschinen).
  - Mit $\max_{j \in \{1,2,\ldots,m\}} \sum_{i \in \{1,2,\ldots,n\} \wedge f(i)=j} p_i$ ist minimal.

- Die Funktion f gibt den Ablaufplan (Schedule) an.

Definition

Definition (Scheduling auf identischen Maschinen)

Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $m \in \mathbb{N}$ (d.h. m identische Maschinen).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$,
    (d.h. Zuweisung der Jobs auf Maschinen).
  - Mit $\max_{j\in\{1,2,\ldots,m\}} \sum_{i\in\{1,2,\ldots,n\}\wedge f(i)=j} p_i$ ist minimal.

- Die Funktion f gibt den Ablaufplan (Schedule) an.
- Da die Jobs keine Fertigstellungszeit (Deadline) haben, ist die Abarbeitungsfolge der Jobs auf einer Maschine beliebig.

Definition

Definition (Scheduling auf identischen Maschinen)

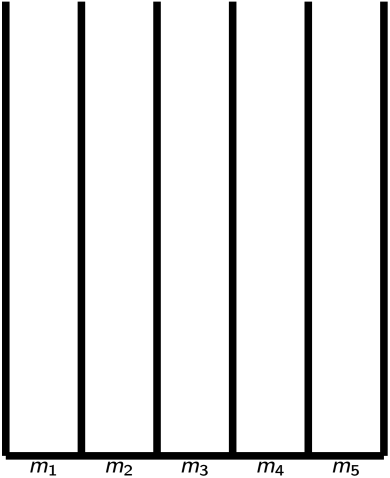
Das Makespan Scheduling Problem auf identischen Maschinen:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $m \in \mathbb{N}$ (d.h. m identische Maschinen).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$,
    (d.h. Zuweisung der Jobs auf Maschinen).
  - Mit $\max_{j \in \{1,2,\ldots,m\}} \sum_{i \in \{1,2,\ldots,n\} \wedge f(i)=j} p_i$ ist minimal.

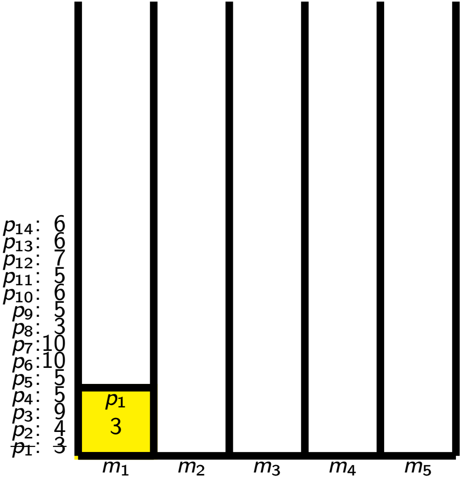
- Die Funktion f gibt den Ablaufplan (Schedule) an.
- Da die Jobs keine Fertigstellungszeit (Deadline) haben, ist die Abarbeitungsfolge der Jobs auf einer Maschine beliebig.
- O.B.d.A.: $n > m$.

Einfaches Beispiel

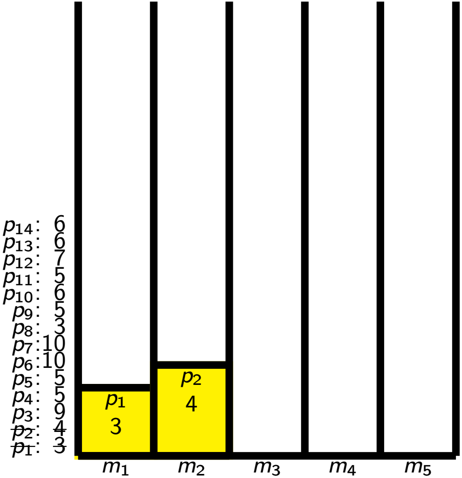
p_{14}: 6
p_{13}: 6
p_{12}: 7
p_{11}: 5
p_{10}: 6
p_9: 5
p_8: 3
p_7: 10
p_6: 10
p_5: 5
p_4: 5
p_3: 9
p_2: 4
p_1: 3



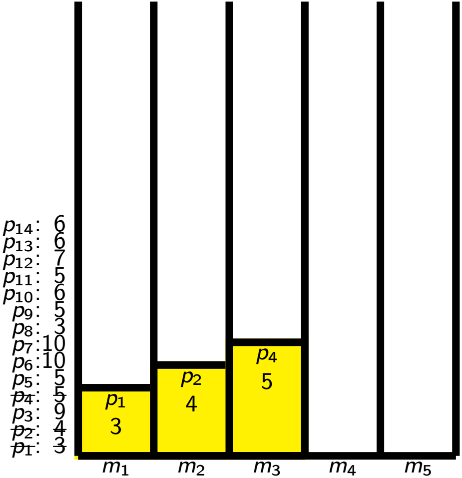
Einfaches Beispiel



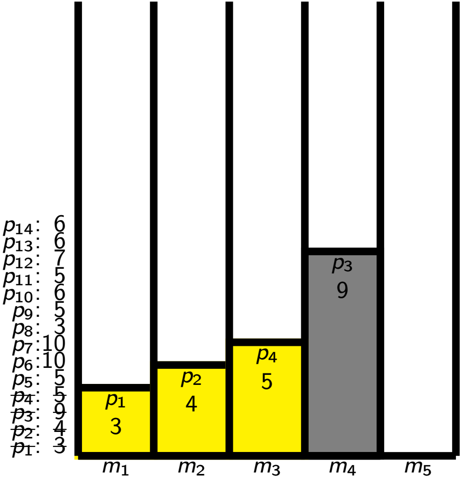
Einfaches Beispiel



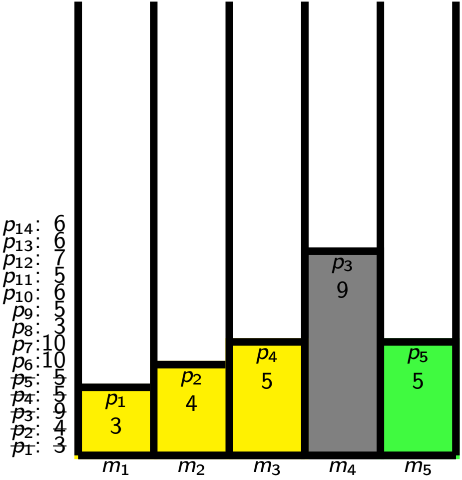
Einfaches Beispiel



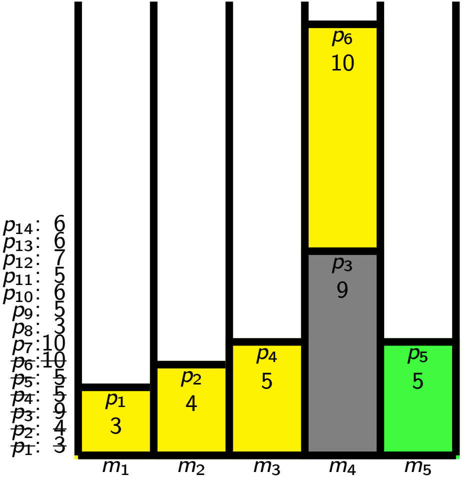
Einfaches Beispiel



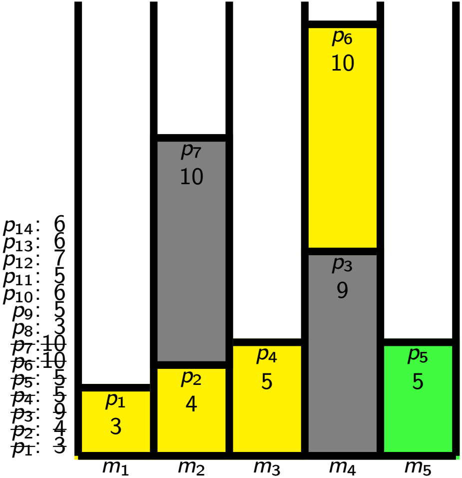
Einfaches Beispiel



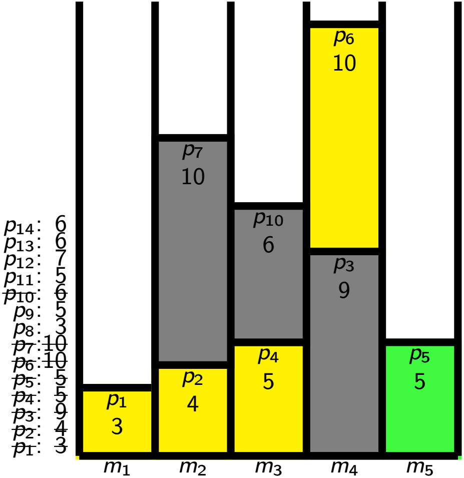
Einfaches Beispiel



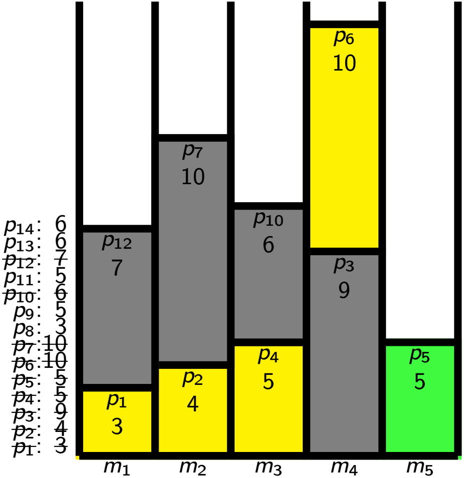
Einfaches Beispiel



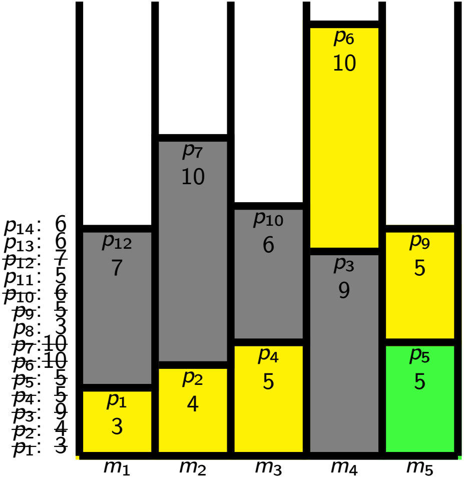
Einfaches Beispiel



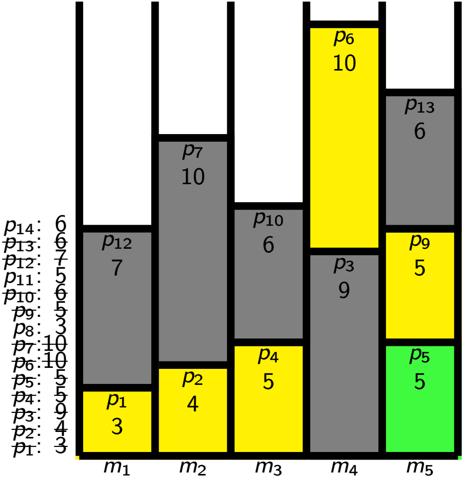
Einfaches Beispiel



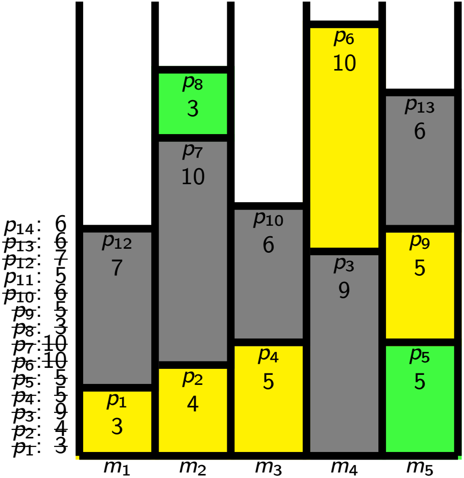
Einfaches Beispiel



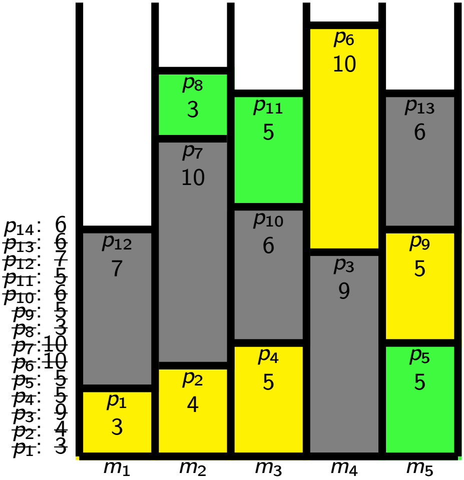
Einfaches Beispiel



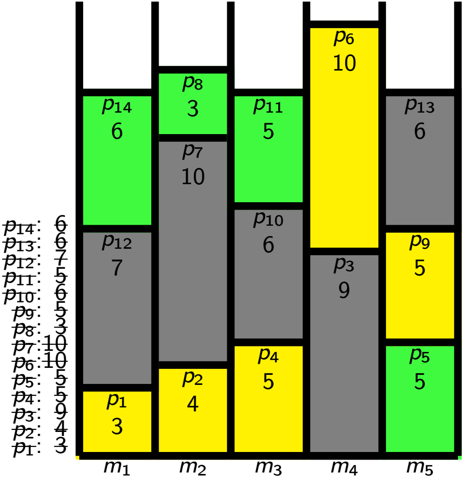
Einfaches Beispiel



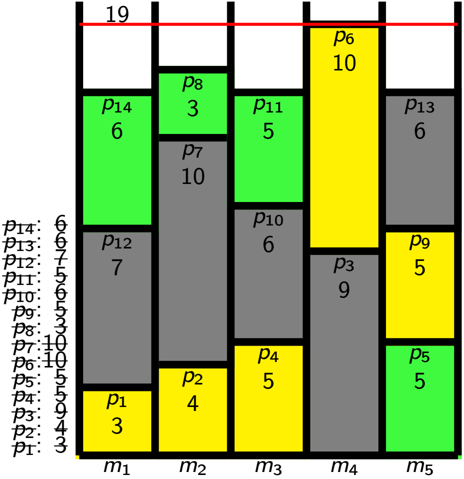
Einfaches Beispiel



Einfaches Beispiel

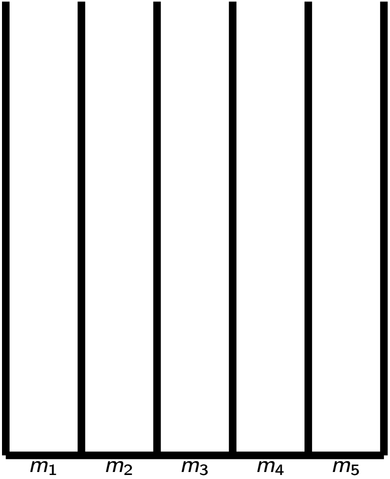


Einfaches Beispiel

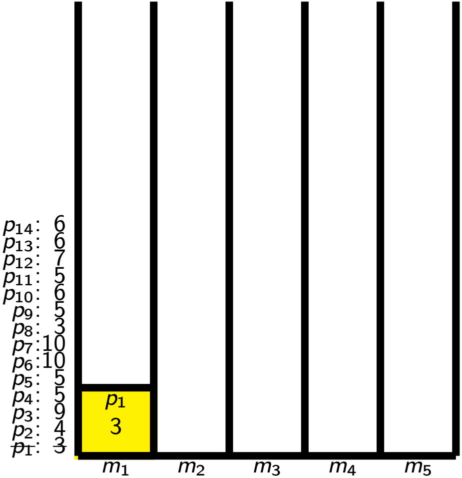


Zweiter Versuch am einfachen Beispiel

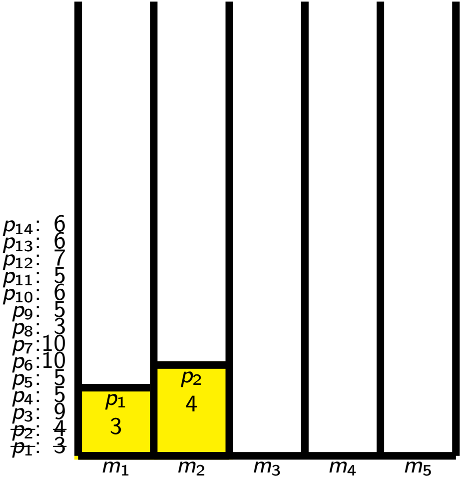
p_{14}: 6
p_{13}: 6
p_{12}: 7
p_{11}: 5
p_{10}: 6
p_9: 5
p_8: 3
p_7: 10
p_6: 10
p_5: 5
p_4: 5
p_3: 9
p_2: 4
p_1: 3



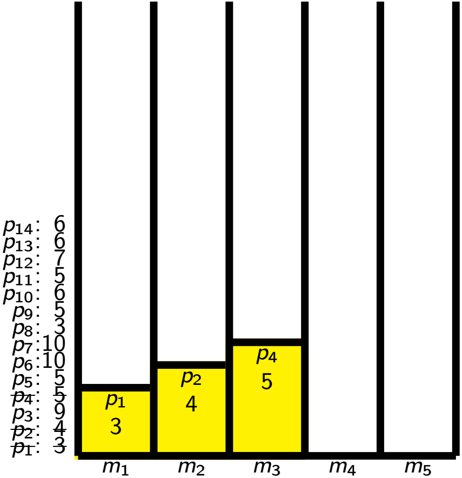
Zweiter Versuch am einfachen Beispiel



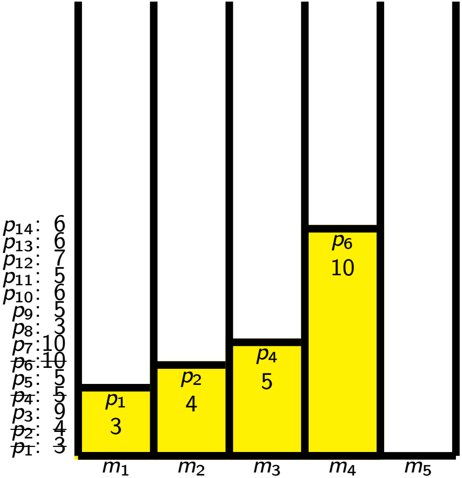
Zweiter Versuch am einfachen Beispiel



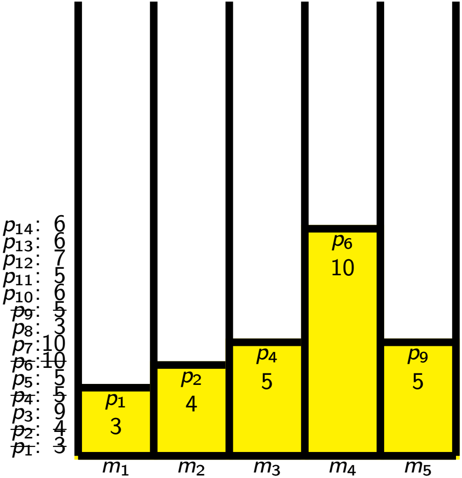
Zweiter Versuch am einfachen Beispiel



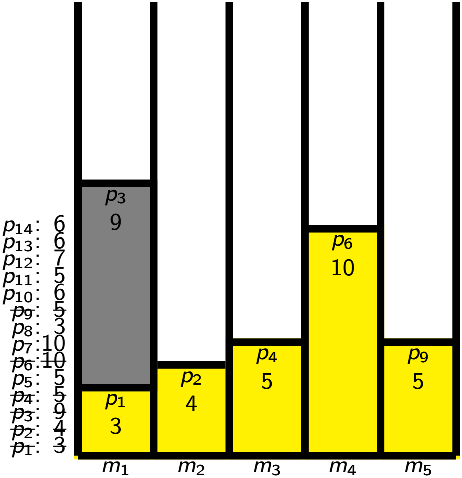
Zweiter Versuch am einfachen Beispiel



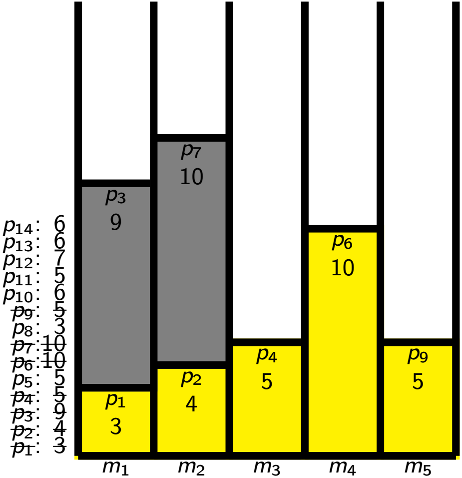
Zweiter Versuch am einfachen Beispiel



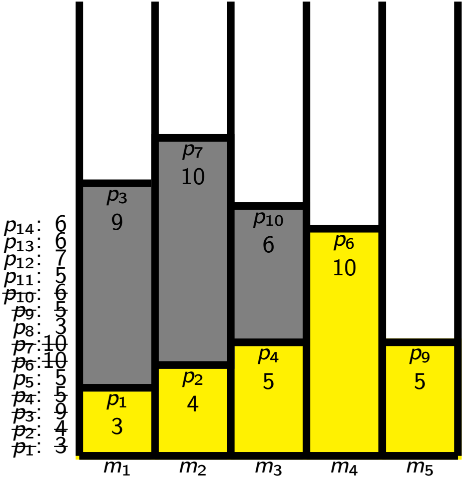
Zweiter Versuch am einfachen Beispiel



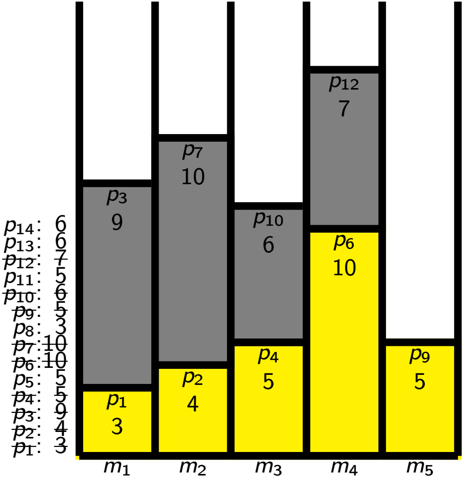
Zweiter Versuch am einfachen Beispiel



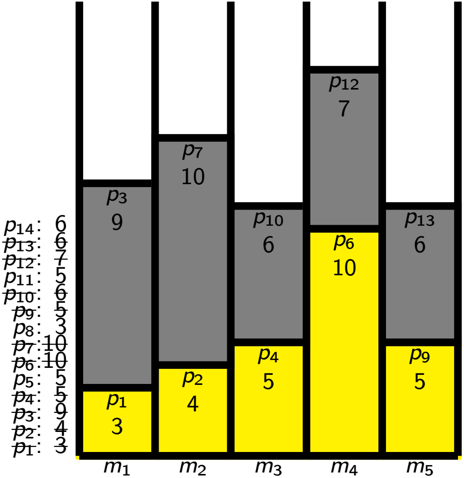
Zweiter Versuch am einfachen Beispiel



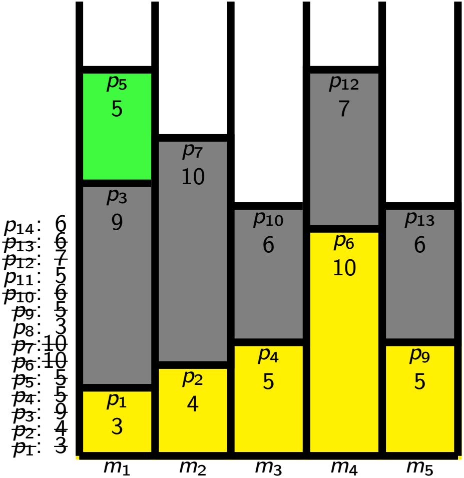
Zweiter Versuch am einfachen Beispiel



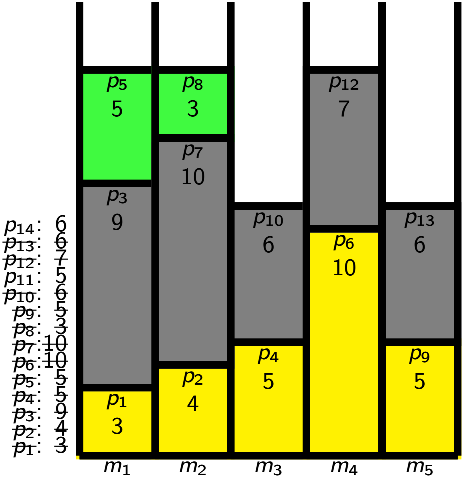
Zweiter Versuch am einfachen Beispiel



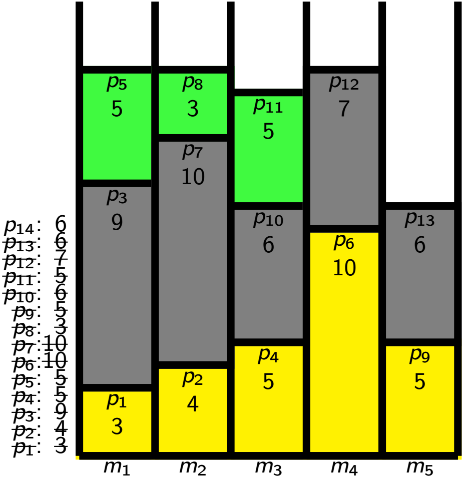
Zweiter Versuch am einfachen Beispiel



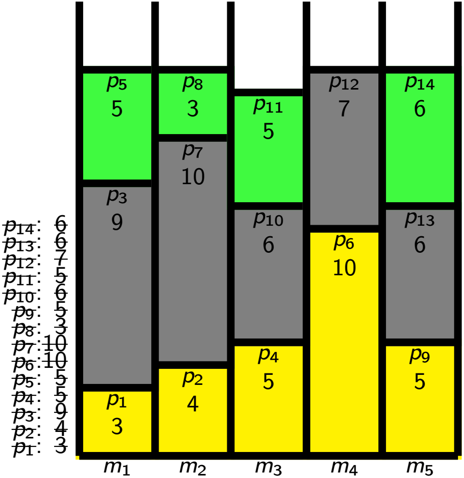
Zweiter Versuch am einfachen Beispiel



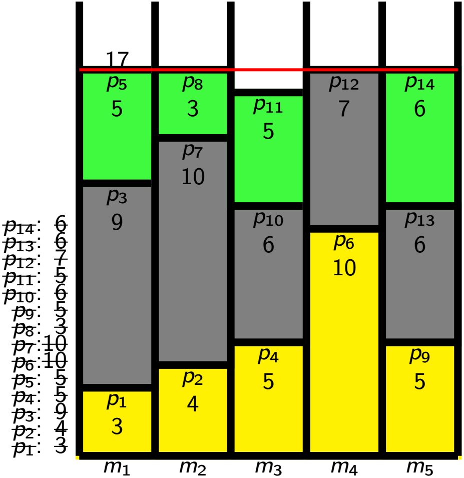
Zweiter Versuch am einfachen Beispiel



Zweiter Versuch am einfachen Beispiel



Zweiter Versuch am einfachen Beispiel



Komplexität

- Das Problem ist in $\mathcal{NPC}$. Einfache Reduktion von Subset-Sum-Problem:

Komplexität

- Das Problem ist in $\mathcal{NPC}$. Einfache Reduktion von Subset-Sum-Problem:
  - Gegeben: b_i für $1 \leqslant i \leqslant n$.

Komplexität

- Das Problem ist in $\mathcal{NPC}$. Einfache Reduktion von Subset-Sum-Problem:
  - Gegeben: b_i für $1 \leqslant i \leqslant n$.
  - Frage: $\exists I \subset \{1, 2, \ldots n\}$ mit: $\sum_{i \in I} b_i = \sum_{i \in \{1,2,\ldots,n\} \setminus I} b_i$.

Komplexität

- Das Problem ist in $\mathcal{NPC}$. Einfache Reduktion von Subset-Sum-Problem:
  - Gegeben: b_i für $1 \leqslant i \leqslant n$.
  - Frage: $\exists I \subset \{1, 2, \ldots n\}$ mit: $\sum_{i \in I} b_i = \sum_{i \in \{1,2,\ldots,n\} \setminus I} b_i$.
- Das ist das Makespan Scheduling Problem für zwei identische Maschinen.

Komplexität

- Das Problem ist in $\mathcal{NPC}$. Einfache Reduktion von Subset-Sum-Problem:
  - Gegeben: b_i für $1 \leqslant i \leqslant n$.
  - Frage: $\exists I \subset \{1, 2, \ldots n\}$ mit: $\sum_{i \in I} b_i = \sum_{i \in \{1,2,\ldots,n\} \setminus I} b_i$.
- Das ist das Makespan Scheduling Problem für zwei identische Maschinen.
- Gesucht ist ein Makespan von $\sum_{i \in \{1,2,\ldots,n\}} b_i/2$.

Heuristik 1

Heuristik: Least Loaded (LL)

Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:

Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:

Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.

Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

Heuristik 1

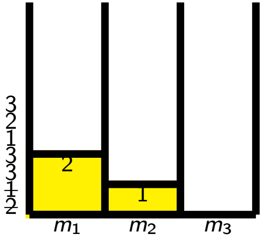
Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i\in\{1,2,\ldots,k-1\}\wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

Heuristik 1

Heuristik: Least Loaded (LL)

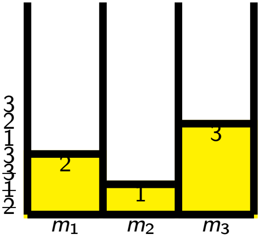
1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

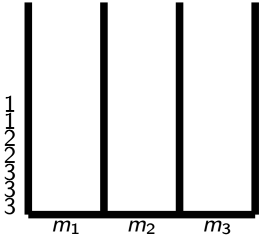


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

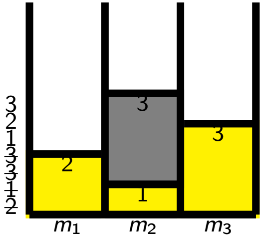


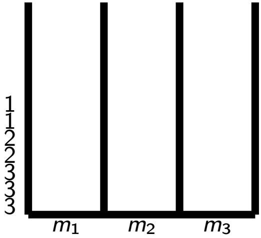


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
    1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
    2. Setze $f(k) = j$.

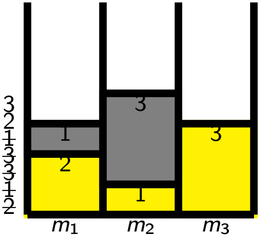


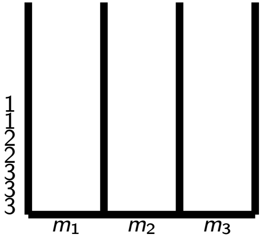


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i\in\{1,2,\ldots,k-1\}\wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

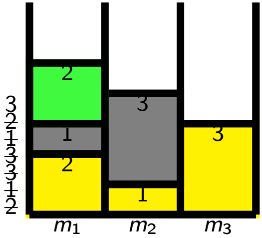


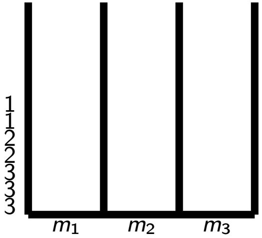


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i\in\{1,2,\ldots,k-1\}\wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

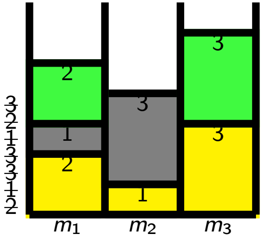


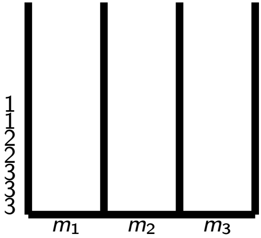


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

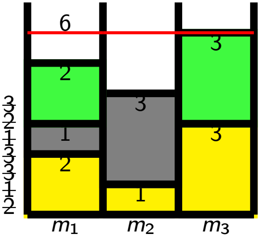


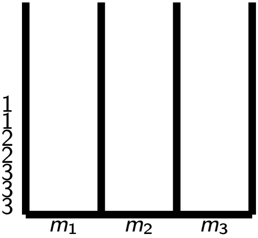


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

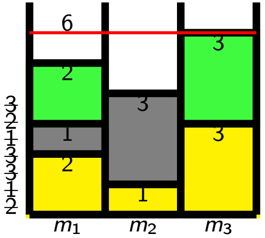


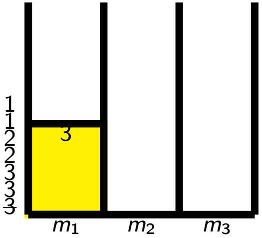


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

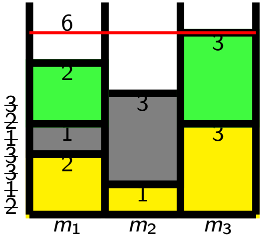


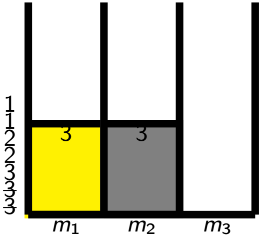


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

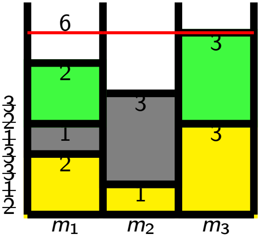


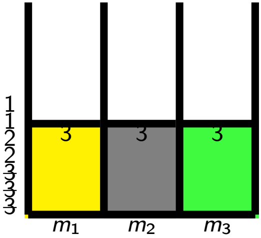


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

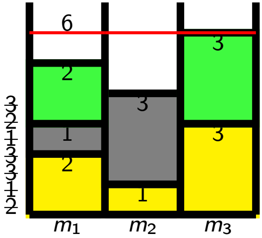


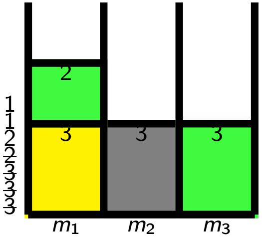


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

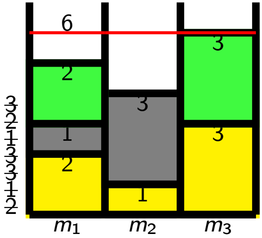


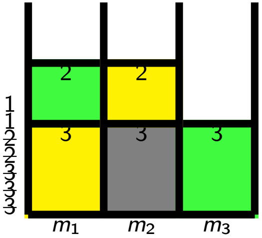


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

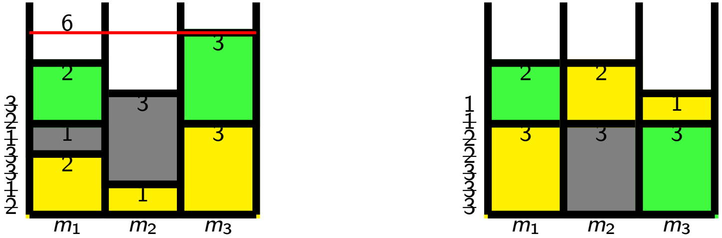




Heuristik 1

Heuristik: Least Loaded (LL)

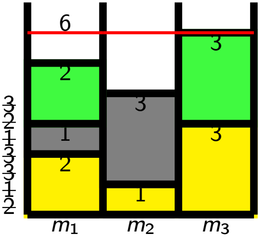
1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

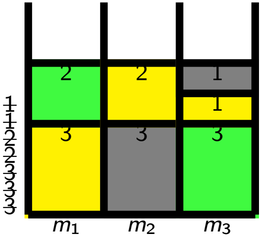


Heuristik 1

Heuristik: Least Loaded (LL)

1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

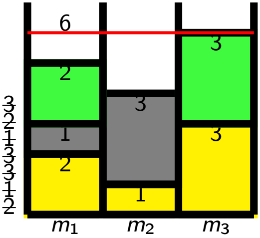


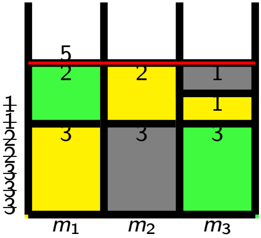


Heuristik 1

Heuristik: Least Loaded (LL)

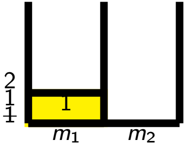
1. Wähle die Maschine, die bisher die kleinste Last hat:
2. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.



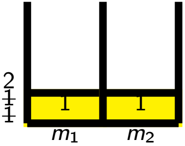


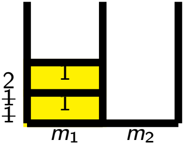
Beispiel zu Heuristik 1 (2 Maschinen)

Beispiel zu Heuristik 1 (2 Maschinen)

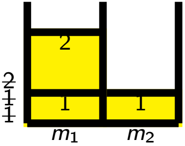


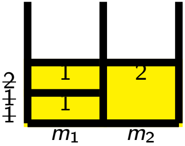
Beispiel zu Heuristik 1 (2 Maschinen)



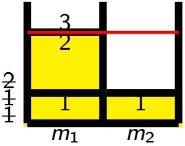


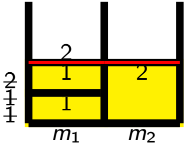
Beispiel zu Heuristik 1 (2 Maschinen)



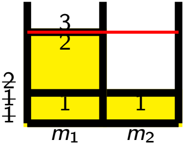


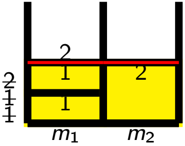
Beispiel zu Heuristik 1 (2,3 Maschinen)



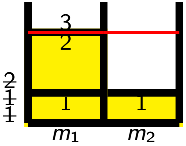


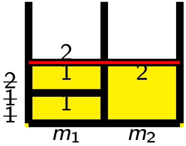
Beispiel zu Heuristik 1 (2,3 Maschinen)

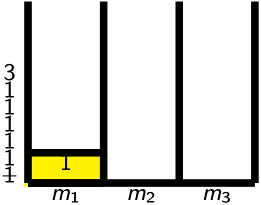




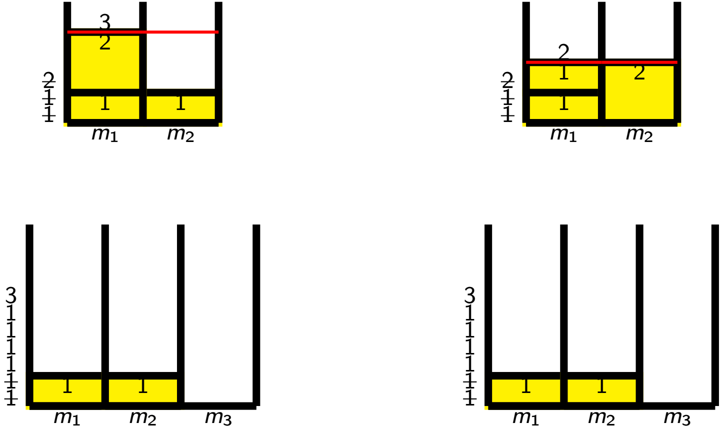
Beispiel zu Heuristik 1 (2,3 Maschinen)



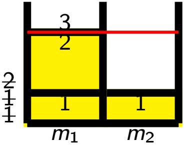


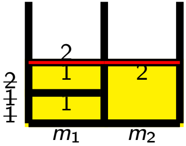


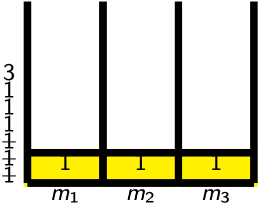
Beispiel zu Heuristik 1 (2,3 Maschinen)

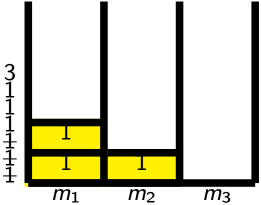


Beispiel zu Heuristik 1 (2,3 Maschinen)

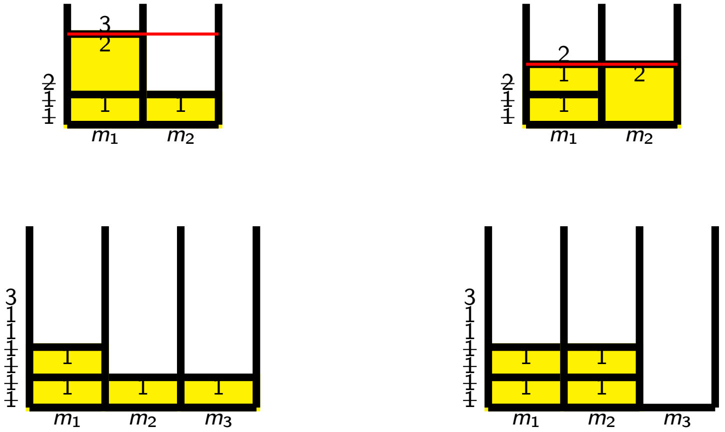




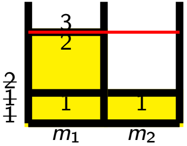


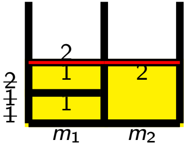


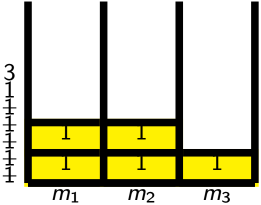
Beispiel zu Heuristik 1 (2,3 Maschinen)

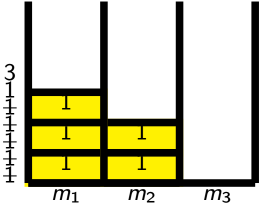


Beispiel zu Heuristik 1 (2,3 Maschinen)

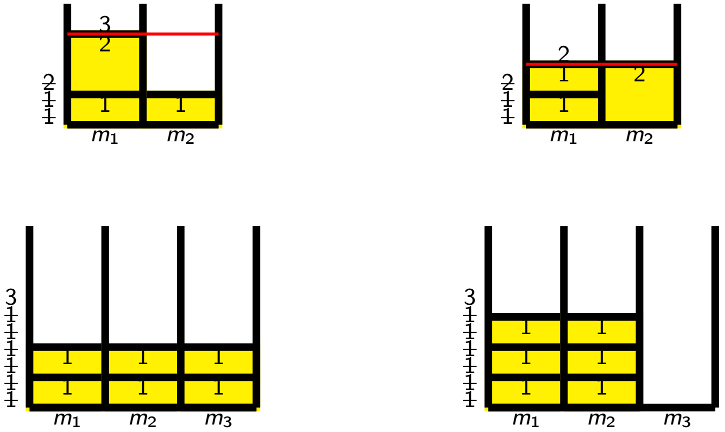




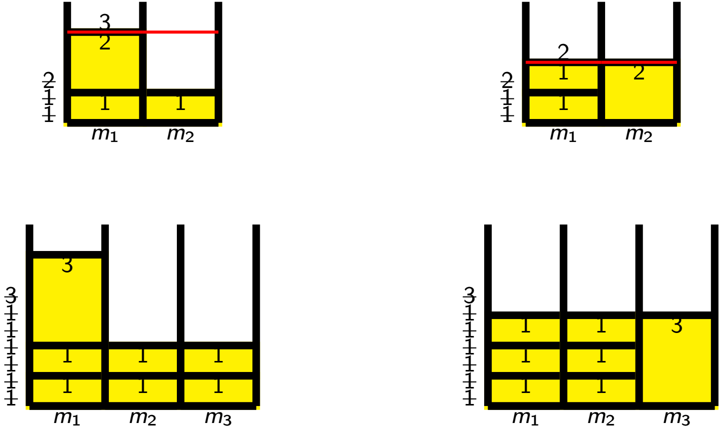




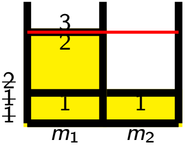
Beispiel zu Heuristik 1 (2,3 Maschinen)

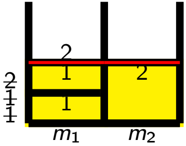


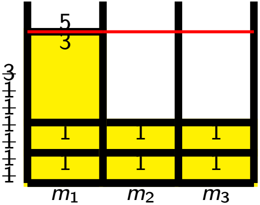
Beispiel zu Heuristik 1 (2,3 Maschinen)

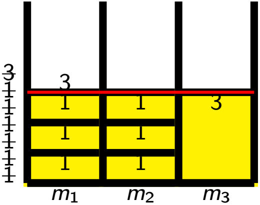


Beispiel zu Heuristik 1 (2,3 Maschinen)



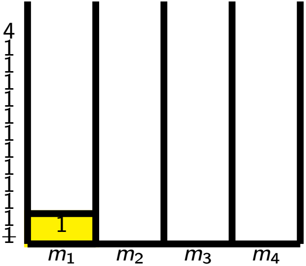




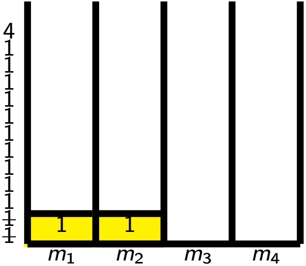


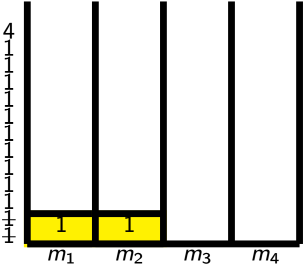
Beispiel zu Heuristik 1 (4 Maschinen)

Beispiel zu Heuristik 1 (4 Maschinen)

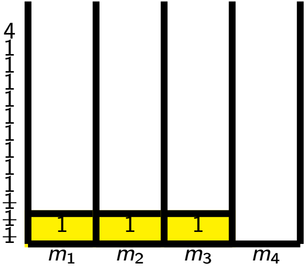


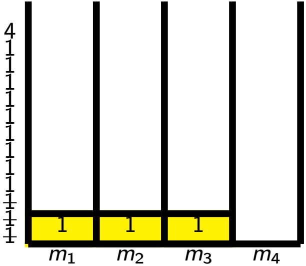
Beispiel zu Heuristik 1 (4 Maschinen)



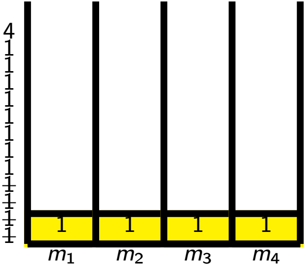


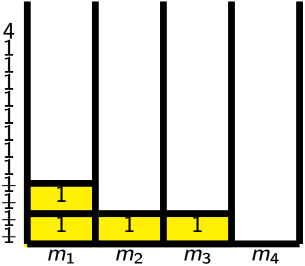
Beispiel zu Heuristik 1 (4 Maschinen)



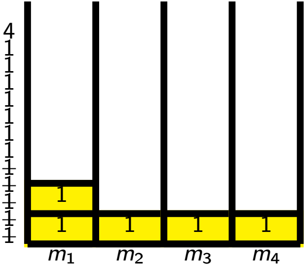


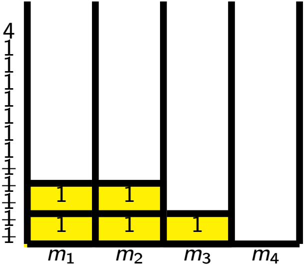
Beispiel zu Heuristik 1 (4 Maschinen)



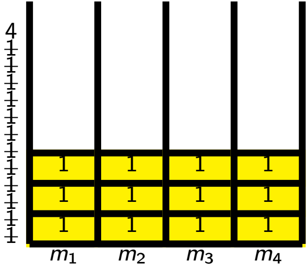


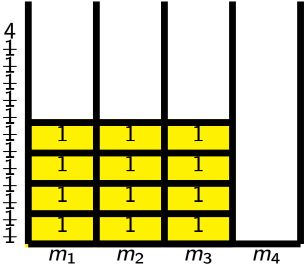
Beispiel zu Heuristik 1 (4 Maschinen)



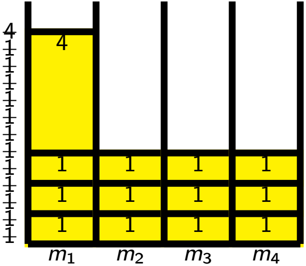


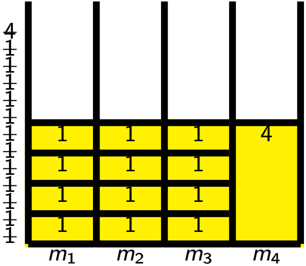
Beispiel zu Heuristik 1 (4 Maschinen)



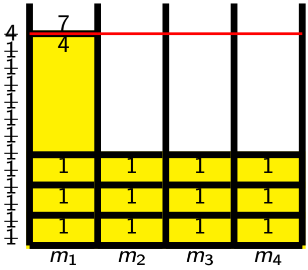


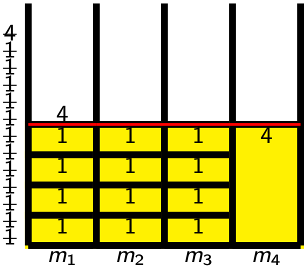
Beispiel zu Heuristik 1 (4 Maschinen)





Beispiel zu Heuristik 1 (4 Maschinen)





Das Beispiel formal

- Konstruiere allgemeines Beispiel:

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.
  - Setze weiter: $p_{m \cdot (m-1)+1} = m$.

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.
  - Setze weiter: $p_{m \cdot (m-1)+1} = m$.
- Die obigen Beispiele zeigen:

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.
  - Setze weiter: $p_{m \cdot (m-1)+1} = m$.
- Die obigen Beispiele zeigen:
  - Optimale Makespan ist m.

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.
  - Setze weiter: $p_{m \cdot (m-1)+1} = m$.
- Die obigen Beispiele zeigen:
  - Optimale Makespan ist m.
    - Auf $m-1$ Maschinen brauchen Jobs $\{1, 2, \ldots, n-1\}$ Zeit m.

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.
  - Setze weiter: $p_{m \cdot (m-1)+1} = m$.
- Die obigen Beispiele zeigen:
  - Optimale Makespan ist m.
    - Auf $m - 1$ Maschinen brauchen Jobs $\{1, 2, \ldots, n-1\}$ Zeit m.
    - Der Job n braucht auf Maschine m auch m Zeit.

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.
  - Setze weiter: $p_{m \cdot (m-1)+1} = m$.
- Die obigen Beispiele zeigen:
  - Optimale Makespan ist m.
    - Auf $m-1$ Maschinen brauchen Jobs $\{1, 2, \ldots, n-1\}$ Zeit m.
    - Der Job n braucht auf Maschine m auch m Zeit.
  - Die LL-Heuristik liefert einen Makespan von $2 \cdot m - 1$.

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.
  - Setze weiter: $p_{m \cdot (m-1)+1} = m$.
- Die obigen Beispiele zeigen:
  - Optimale Makespan ist m.
    - Auf $m-1$ Maschinen brauchen Jobs $\{1, 2, \ldots, n-1\}$ Zeit m.
    - Der Job n braucht auf Maschine m auch m Zeit.
  - Die LL-Heuristik liefert einen Makespan von $2 \cdot m - 1$.
    - Auf m Maschinen brauchen $\{1, 2, \ldots, n-1\}$ Zeit $m-1$.

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.
  - Setze weiter: $p_{m \cdot (m-1)+1} = m$.
- Die obigen Beispiele zeigen:
  - Optimale Makespan ist m.
    - Auf $m-1$ Maschinen brauchen Jobs $\{1, 2, \ldots, n-1\}$ Zeit m.
    - Der Job n braucht auf Maschine m auch m Zeit.
  - Die LL-Heuristik liefert einen Makespan von $2 \cdot m - 1$.
    - Auf m Maschinen brauchen $\{1, 2, \ldots, n-1\}$ Zeit $m-1$.
    - Der Job n braucht auf einer Maschine noch zusätzlich m Zeit.

Das Beispiel formal

- Konstruiere allgemeines Beispiel:
  - Setze $n = m \cdot (m-1) + 1$.
  - Für $i \in \{1, 2, \ldots, m \cdot (m-1)\}$ setze: $p_i = 1$.
  - Setze weiter: $p_{m \cdot (m-1)+1} = m$.
- Die obigen Beispiele zeigen:
  - Optimale Makespan ist m.
    - Auf $m-1$ Maschinen brauchen Jobs $\{1, 2, \ldots, n-1\}$ Zeit m.
    - Der Job n braucht auf Maschine m auch m Zeit.
  - Die LL-Heuristik liefert einen Makespan von $2 \cdot m - 1$.
    - Auf m Maschinen brauchen $\{1, 2, \ldots, n-1\}$ Zeit $m-1$.
    - Der Job n braucht auf einer Maschine noch zusätzlich m Zeit.
- Damit ist der Approximationsfaktor von der LL-Heuristik bestenfalls:

$$\frac{2 \cdot m - 1}{m} = 2 - \frac{1}{m}.$$

Approximationsgüte von Heuristik 1

Theorem

Die LL-Heuristik hat einen Approximationsfaktor von $(2 - 1/m)$.

Beweis:

Approximationsgüte von Heuristik 1

Theorem

Die LL-Heuristik hat einen Approximationsfaktor von $(2 - 1/m)$.

Beweis:

Approximationsgüte von Heuristik 1

Theorem

Die LL-Heuristik hat einen Approximationsfaktor von $(2 - 1/m)$.

Beweis:

Approximationsgüte von Heuristik 1

Theorem

Die LL-Heuristik hat einen Approximationsfaktor von $(2 - 1/m)$.

Beweis:

- Die folgenden einfachen Schranken gelten:

Approximationsgüte von Heuristik 1

Theorem

Die LL-Heuristik hat einen Approximationsfaktor von $(2 - 1/m)$.

Beweis:

- Die folgenden einfachen Schranken gelten:
  - $opt \geqslant \max_{i \in \{1,2,\ldots,n\}}(p_i)$.

Approximationsgüte von Heuristik 1

Theorem

Die LL-Heuristik hat einen Approximationsfaktor von $(2 - 1/m)$.

Beweis:

- Die folgenden einfachen Schranken gelten:
  - $opt \geqslant \max_{i\in\{1,2,\ldots,n\}}(p_i)$.
  - $opt \geqslant \frac{1}{m} \cdot \sum_{i\in\{1,2,\ldots,n\}} p_i$.

Approximationsgüte von Heuristik 1

Theorem

Die LL-Heuristik hat einen Approximationsfaktor von $(2 - 1/m)$.

Beweis:

- Die folgenden einfachen Schranken gelten:
  - $opt \geqslant \max_{i \in \{1,2,\ldots,n\}}(p_i)$.
  - $opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$.
- Betrachte nun den Job, der als letzter fertig wird und

Approximationsgüte von Heuristik 1

Theorem

Die LL-Heuristik hat einen Approximationsfaktor von $(2 - 1/m)$.

Beweis:

- Die folgenden einfachen Schranken gelten:
  - $opt \geqslant \max_{i \in \{1,2,\ldots,n\}}(p_i)$.
  - $opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$.
- Betrachte nun den Job, der als letzter fertig wird und
- die Situation, bevor er verteilt wurde.

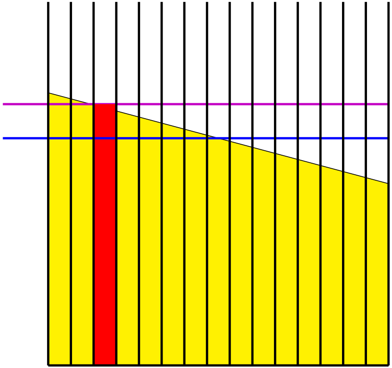
Approximationsgüte von Heuristik 1

Theorem

Die LL-Heuristik hat einen Approximationsfaktor von $(2 - 1/m)$.

Beweis:

- Die folgenden einfachen Schranken gelten:
  - $opt \geqslant \max_{i \in \{1,2,\ldots,n\}}(p_i)$.
  - $opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$.
- Betrachte nun den Job, der als letzter fertig wird und
- die Situation, bevor er verteilt wurde.



Approximationsgüte von Heuristik 1

$opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$
$opt \geqslant \max_{i \in \{1,2,\ldots,n\}} (p_i)$

Approximationsgüte von Heuristik 1

$opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$
$opt \geqslant \max_{i \in \{1,2,\ldots,n\}}(p_i)$

Approximationsgüte von Heuristik 1

$opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,...,n\}} p_i$
$opt \geqslant \max_{i \in \{1,2,...,n\}}(p_i)$

- Sei i' der Job, der als letzter fertig wird und $j' = f(i')$.

Approximationsgüte von Heuristik 1

$opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$
$opt \geqslant \max_{i \in \{1,2,\ldots,n\}} (p_i)$

- Sei i' der Job, der als letzter fertig wird und $j' = f(i')$.
- Als der Job i' auf j' gelegt wurde, war aktuelle Last von j' minimal.

Approximationsgüte von Heuristik 1

$opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$
$opt \geqslant \max_{i \in \{1,2,\ldots,n\}} (p_i)$

- Sei i' der Job, der als letzter fertig wird und $j' = f(i')$.
- Als der Job i' auf j' gelegt wurde, war aktuelle Last von j' minimal.
- Die Last war höchstens: $1/m \sum_{i \in \{1,2,\ldots,i'-1\}} p_i$.

Approximationsgüte von Heuristik 1

$opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$
$opt \geqslant \max_{i \in \{1,2,\ldots,n\}}(p_i)$

- Sei i' der Job, der als letzter fertig wird und $j' = f(i')$.
- Als der Job i' auf j' gelegt wurde, war aktuelle Last von j' minimal.
- Die Last war höchstens: $1/m \sum_{i \in \{1,2,\ldots,i'-1\}} p_i$.
- Damit kann der Makespan wie folgt abgeschätzt werden:

$$p_{i'} + \frac{1}{m} \sum_{i \in \{1,2,\ldots,i'-1\}} p_i$$

$$=$$

$$\leqslant$$

$$\leqslant$$

Approximationsgüte von Heuristik 1

$opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$
$opt \geqslant \max_{i \in \{1,2,\ldots,n\}} (p_i)$

- Sei i' der Job, der als letzter fertig wird und $j' = f(i')$.
- Als der Job i' auf j' gelegt wurde, war aktuelle Last von j' minimal.
- Die Last war höchstens: $1/m \sum_{i \in \{1,2,\ldots,i'-1\}} p_i$.
- Damit kann der Makespan wie folgt abgeschätzt werden:

$$
\begin{array}{rl}
 & p_{i'} + \frac{1}{m} \sum_{i \in \{1,2,\ldots,i'-1\}} p_i \\
= & (1 - \frac{1}{m}) \cdot p_{i'} + \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,i'\}} p_i \\
\leqslant & \\
\leqslant &
\end{array}
$$

Approximationsgüte von Heuristik 1

$opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$
$opt \geqslant \max_{i \in \{1,2,\ldots,n\}} (p_i)$

- Sei i' der Job, der als letzter fertig wird und $j' = f(i')$.
- Als der Job i' auf j' gelegt wurde, war aktuelle Last von j' minimal.
- Die Last war höchstens: $1/m \sum_{i \in \{1,2,\ldots,i'-1\}} p_i$.
- Damit kann der Makespan wie folgt abgeschätzt werden:

$$
\begin{aligned}
& p_{i'} + \frac{1}{m} \sum_{i \in \{1,2,\ldots,i'-1\}} p_i \\
= \quad & (1 - \frac{1}{m}) \cdot p_{i'} + \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,i'\}} p_i \\
\leqslant \quad & (1 - \frac{1}{m}) \cdot opt + opt \\
\leqslant \quad &
\end{aligned}
$$

Approximationsgüte von Heuristik 1

$opt \geqslant \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,n\}} p_i$
$opt \geqslant \max_{i \in \{1,2,\ldots,n\}} (p_i)$

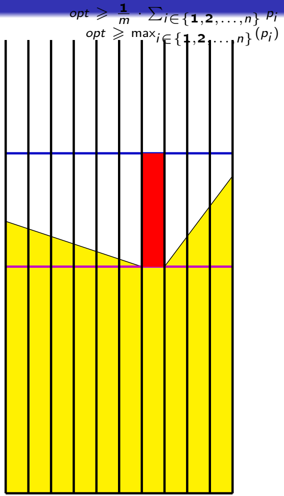
- Sei i' der Job, der als letzter fertig wird und $j' = f(i')$.
- Als der Job i' auf j' gelegt wurde, war aktuelle Last von j' minimal.
- Die Last war höchstens: $1/m \sum_{i \in \{1,2,\ldots,i'-1\}} p_i$.
- Damit kann der Makespan wie folgt abgeschätzt werden:

$$\begin{aligned} & p_{i'} + \frac{1}{m} \sum_{i \in \{1,2,\ldots,i'-1\}} p_i \\ = \quad & (1 - \frac{1}{m}) \cdot p_{i'} + \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,i'\}} p_i \\ \leqslant \quad & (1 - \frac{1}{m}) \cdot opt + opt \\ \leqslant \quad & (2 - \frac{1}{m}) \cdot opt. \end{aligned}$$

Approximationsgüte von Heuristik 1

- Sei i' der Job, der als letzter fertig wird und $j' = f(i')$.
- Als der Job i' auf j' gelegt wurde, war aktuelle Last von j' minimal.
- Die Last war höchstens: $1/m \sum_{i \in \{1,2,\ldots,i'-1\}} p_i$.
- Damit kann der Makespan wie folgt abgeschätzt werden:

$$\begin{aligned} & p_{i'} + \frac{1}{m} \sum_{i \in \{1,2,\ldots,i'-1\}} p_i \\ = \quad & (1 - \tfrac{1}{m}) \cdot p_{i'} + \frac{1}{m} \cdot \sum_{i \in \{1,2,\ldots,i'\}} p_i \\ \leqslant \quad & (1 - \tfrac{1}{m}) \cdot opt + opt \\ \leqslant \quad & (2 - \tfrac{1}{m}) \cdot opt. \end{aligned}$$



Heuristik 2

Heuristik: Longest Processing Time (LPT)

Heuristik 2

Heuristik: Longest Processing Time (LPT)

1. Wähle für den längsten Job die Maschine, die bisher die kleinste Last hat:

Heuristik 2

Heuristik: Longest Processing Time (LPT)

1. Wähle für den längsten Job die Maschine, die bisher die kleinste Last hat:
2. Sei $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.

Heuristik 2

Heuristik: Longest Processing Time (LPT)

1. Wähle für den längsten Job die Maschine, die bisher die kleinste Last hat:
2. Sei $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.
3. Für k von 1 bis n mache:

Heuristik 2

Heuristik: Longest Processing Time (LPT)

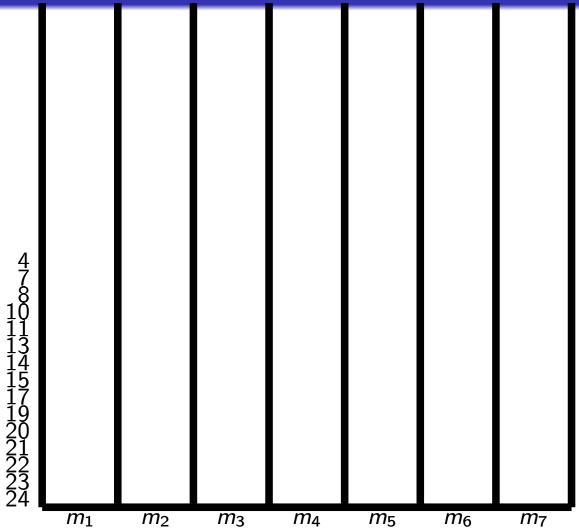
1. Wähle für den längsten Job die Maschine, die bisher die kleinste Last hat:
2. Sei $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.
3. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.

Heuristik 2

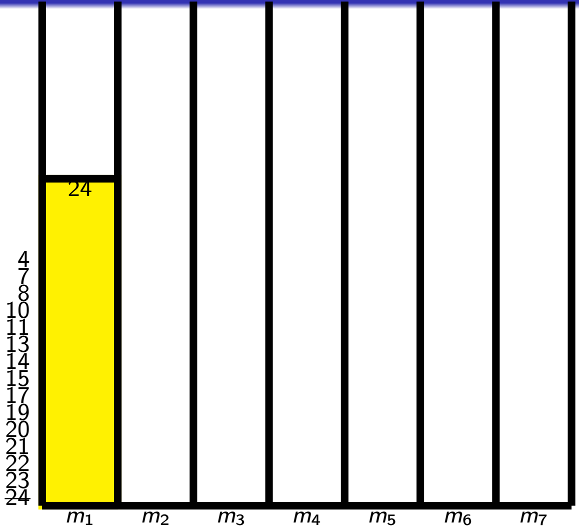
Heuristik: Longest Processing Time (LPT)

1. Wähle für den längsten Job die Maschine, die bisher die kleinste Last hat:
2. Sei $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.
3. Für k von 1 bis n mache:
   1. Wähle j mit $\sum_{i \in \{1,2,\ldots,k-1\} \wedge f(i)=j} p_i$ ist minimal.
   2. Setze $f(k) = j$.

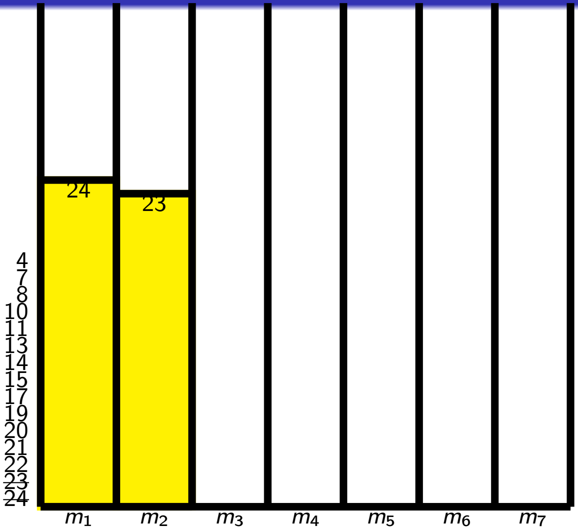
Beispiel zu Heuristik 2



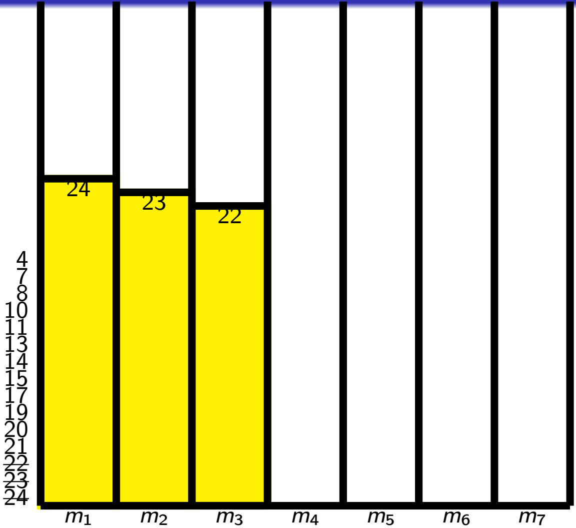
Beispiel zu Heuristik 2



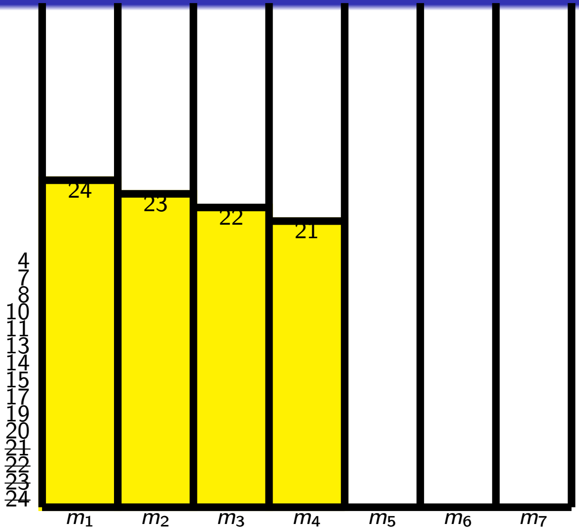
Beispiel zu Heuristik 2



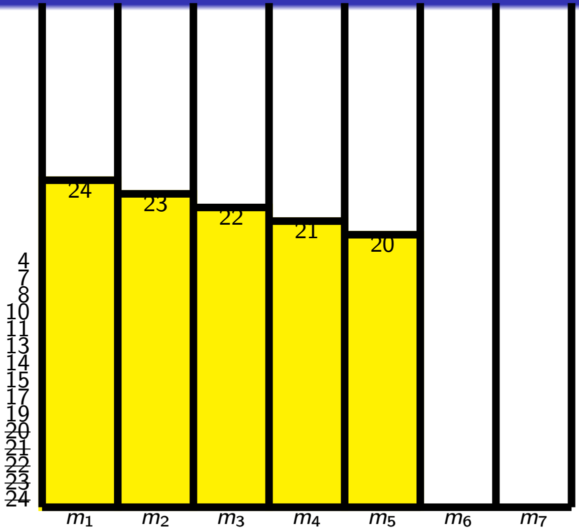
Beispiel zu Heuristik 2



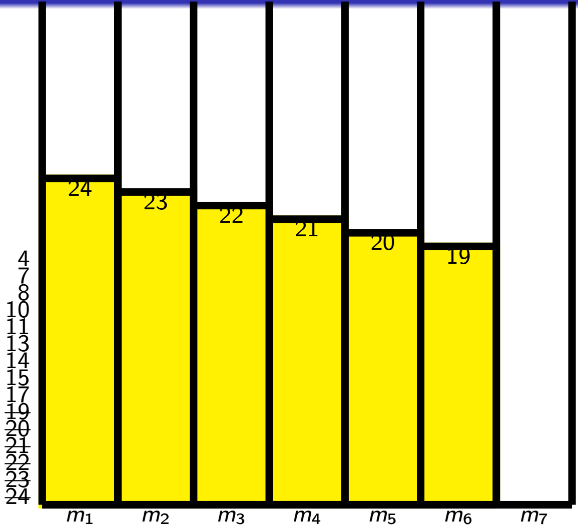
Beispiel zu Heuristik 2



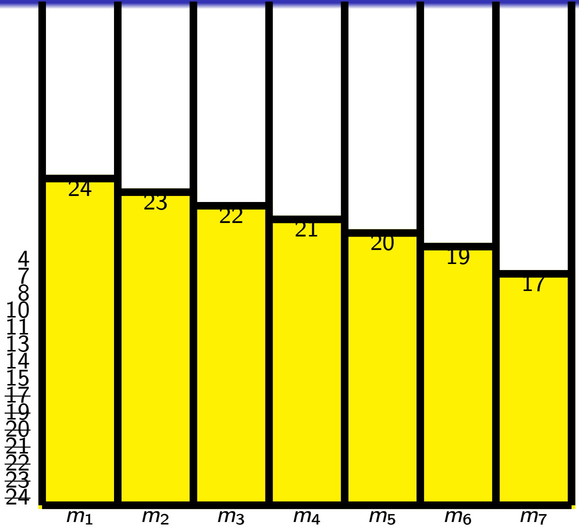
Beispiel zu Heuristik 2



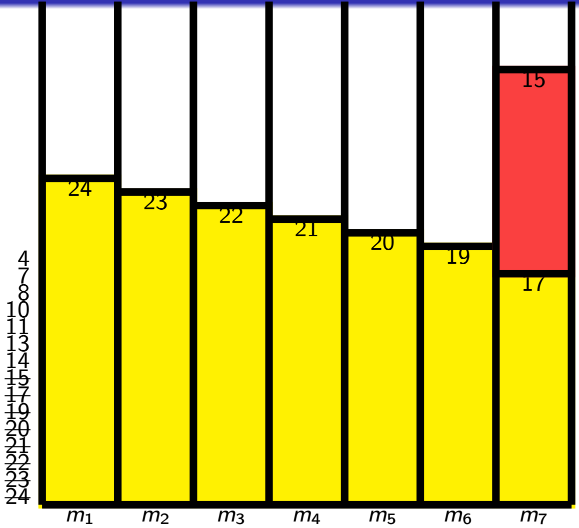
Beispiel zu Heuristik 2



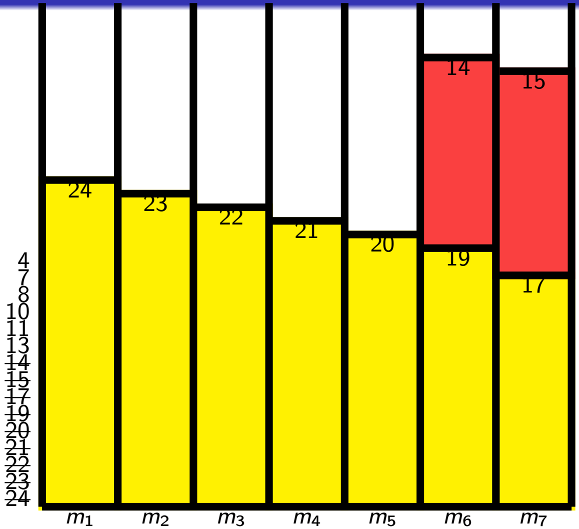
Beispiel zu Heuristik 2



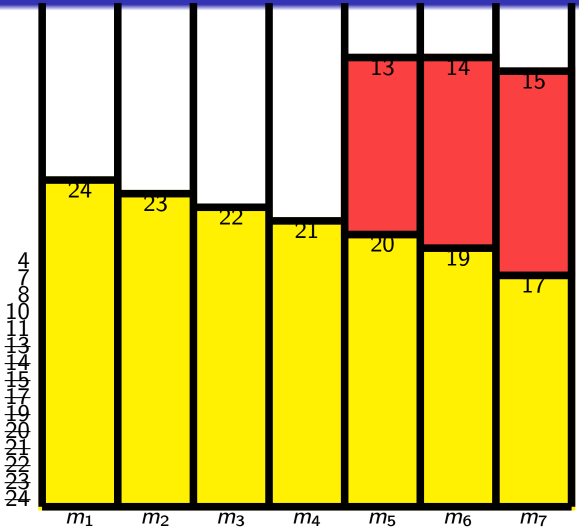
Beispiel zu Heuristik 2



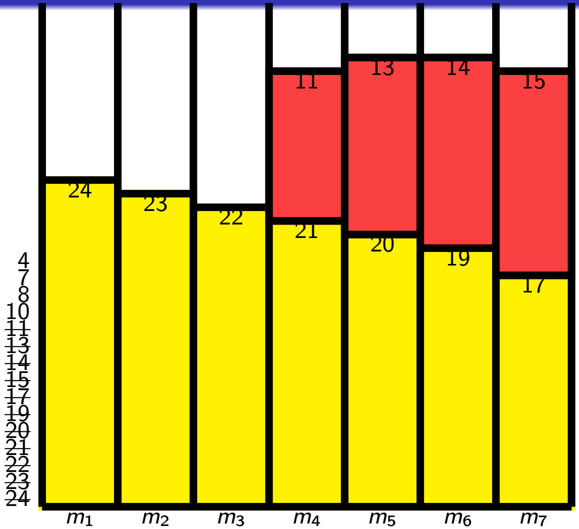
Beispiel zu Heuristik 2



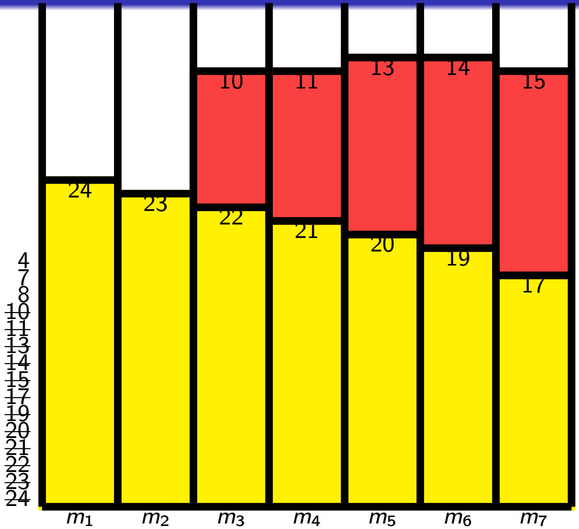
Beispiel zu Heuristik 2



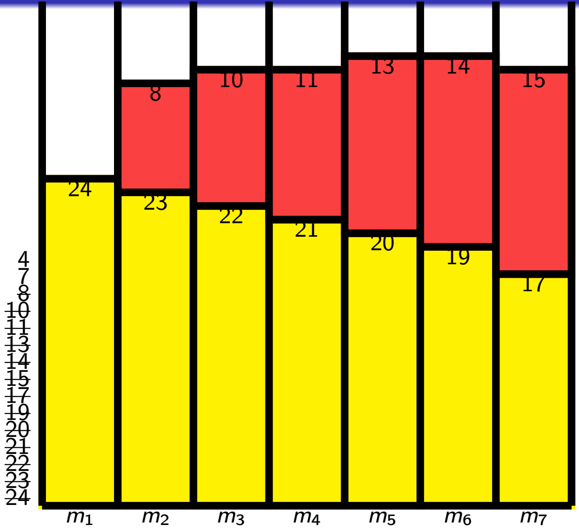
Beispiel zu Heuristik 2



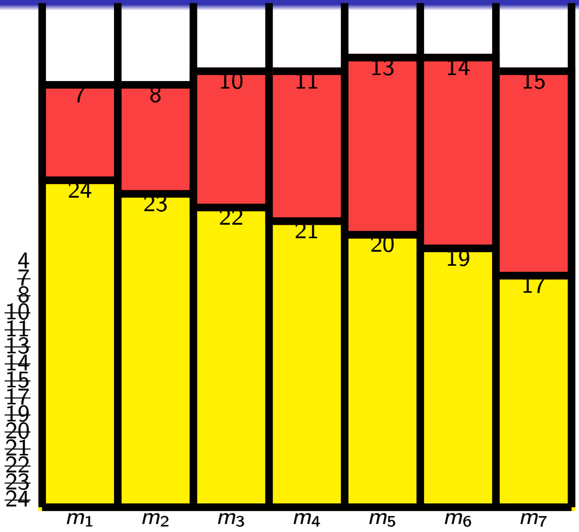
Beispiel zu Heuristik 2



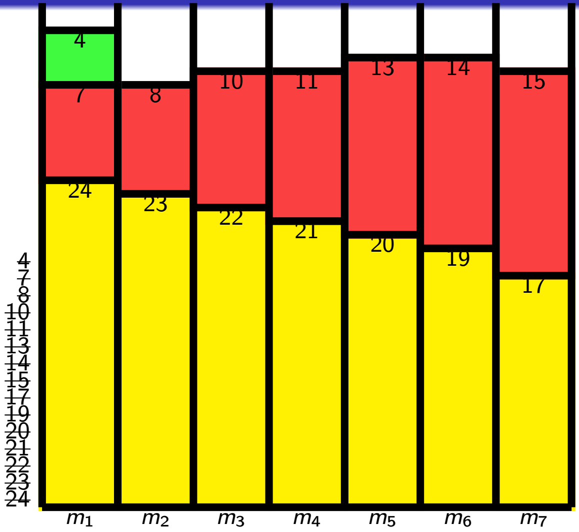
Beispiel zu Heuristik 2



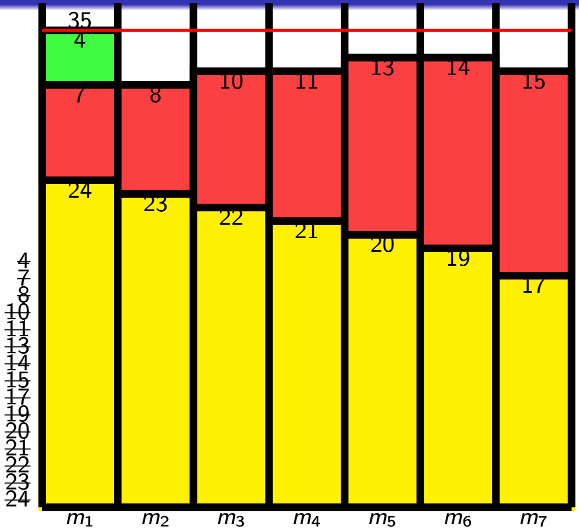
Beispiel zu Heuristik 2



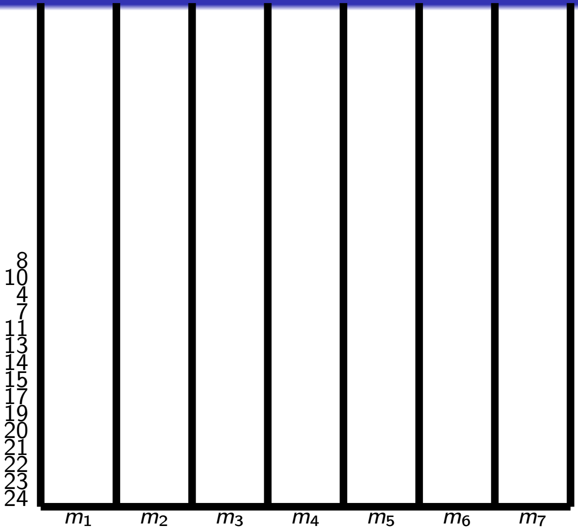
Beispiel zu Heuristik 2



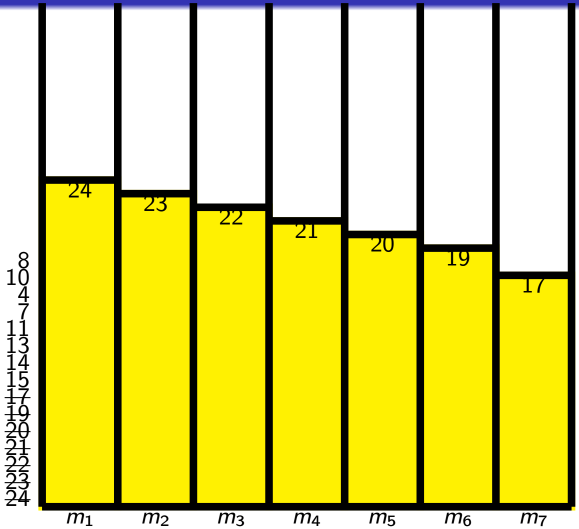
Beispiel zu Heuristik 2



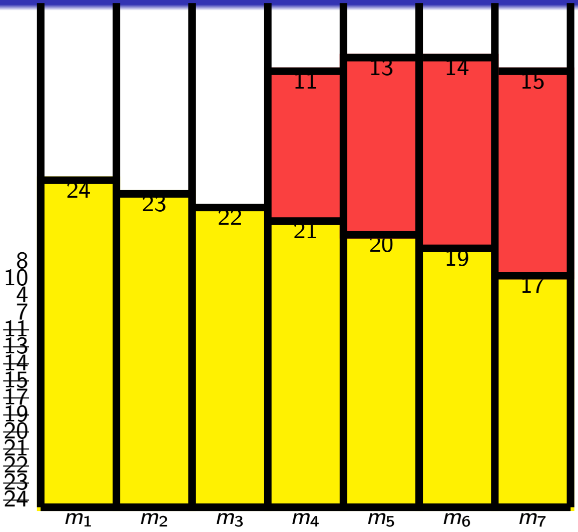
Bessere Lösung des Beispiels



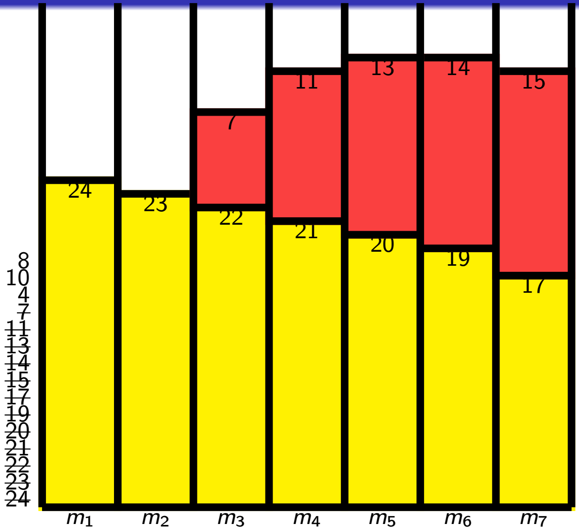
Bessere Lösung des Beispiels



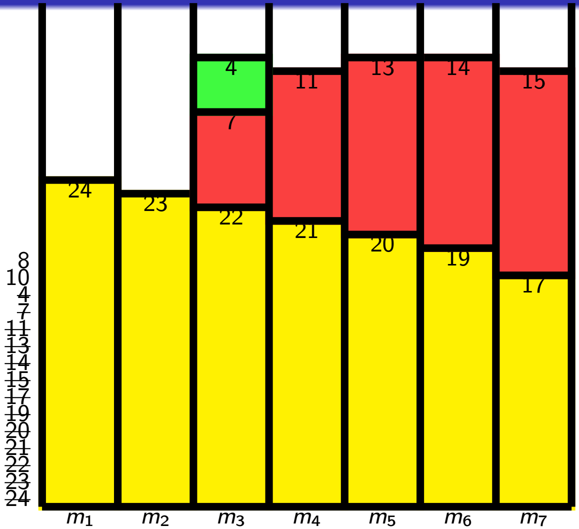
Bessere Lösung des Beispiels



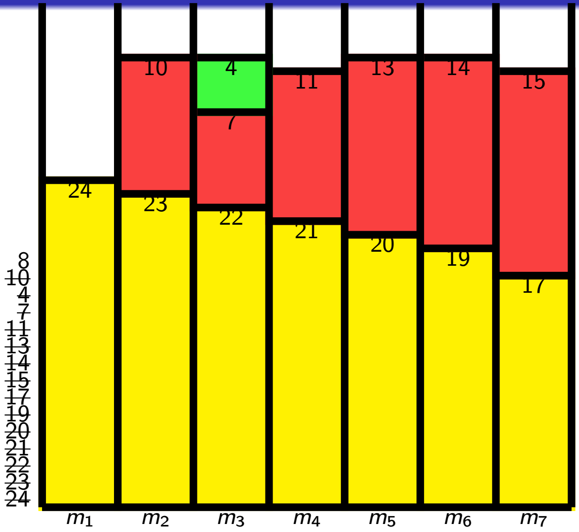
Bessere Lösung des Beispiels



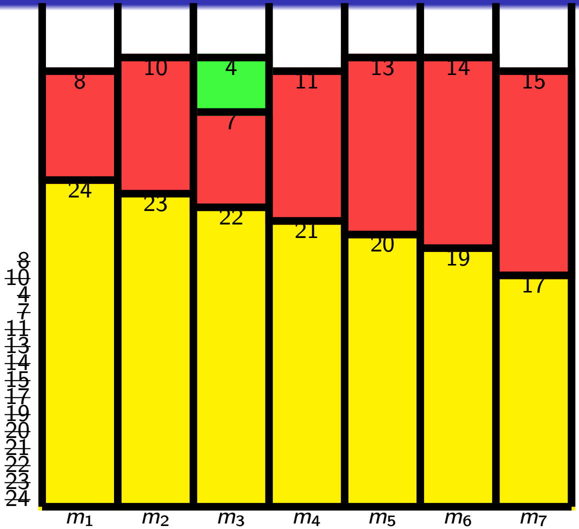
Bessere Lösung des Beispiels



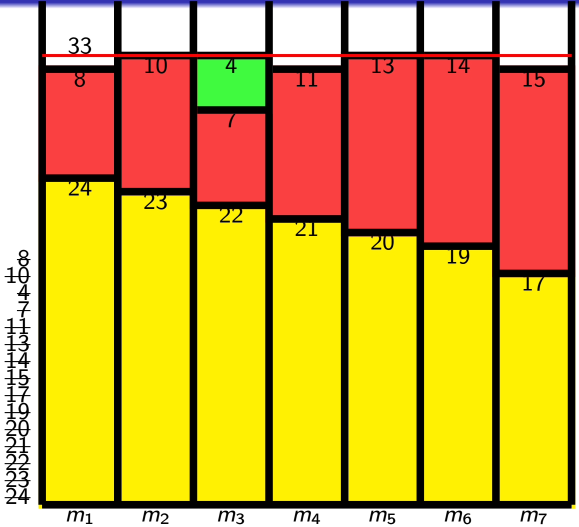
Bessere Lösung des Beispiels



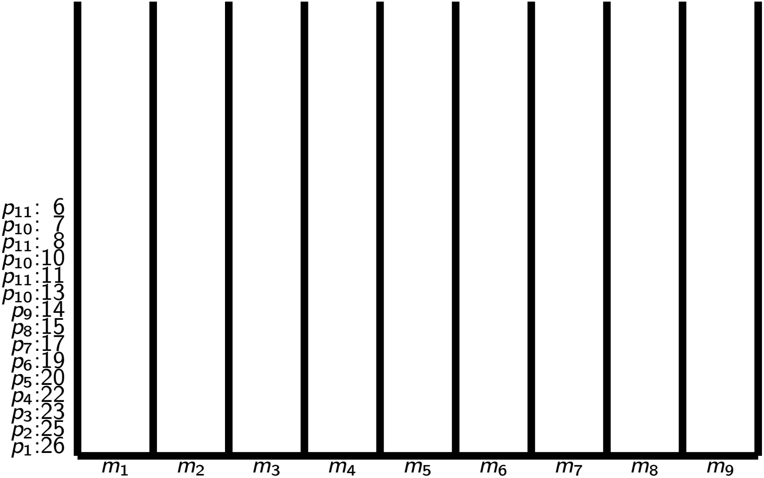
Bessere Lösung des Beispiels



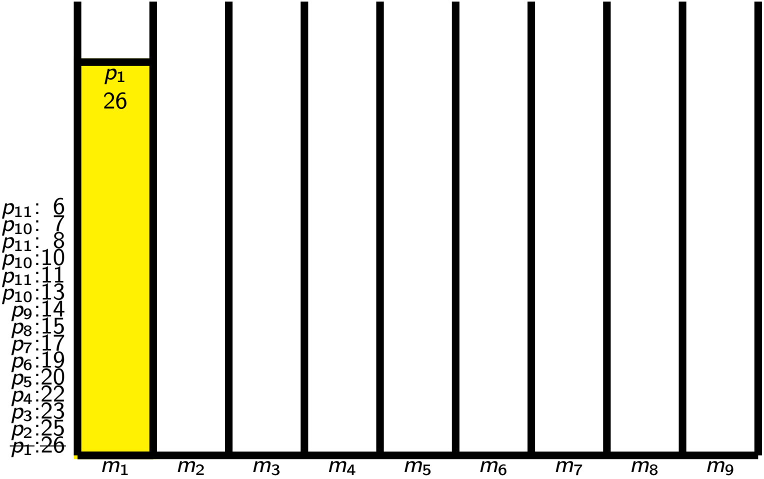
Bessere Lösung des Beispiels



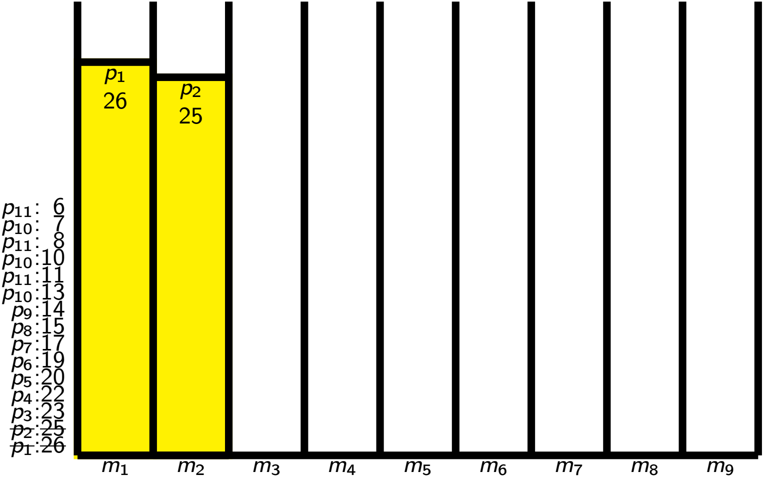
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



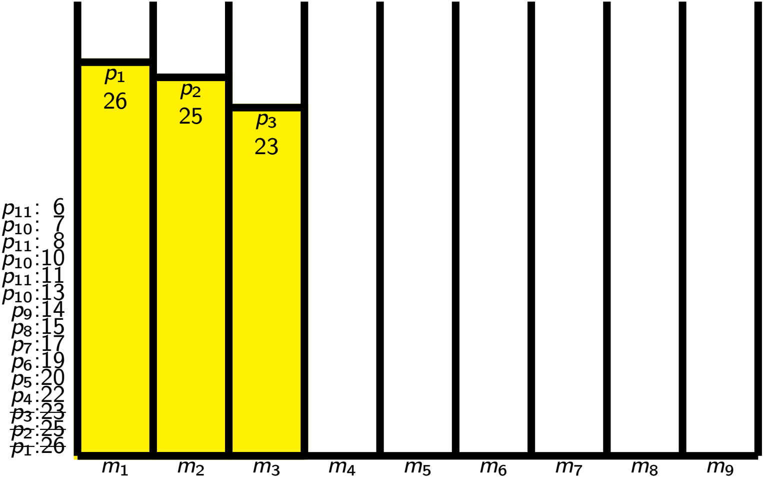
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



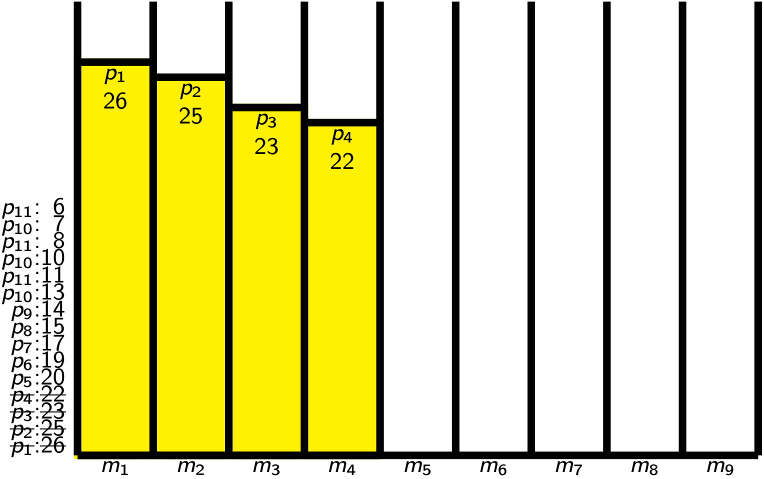
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



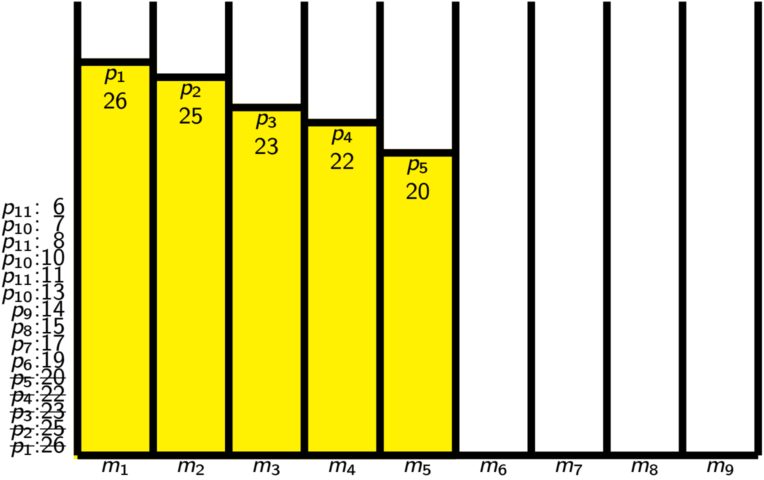
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



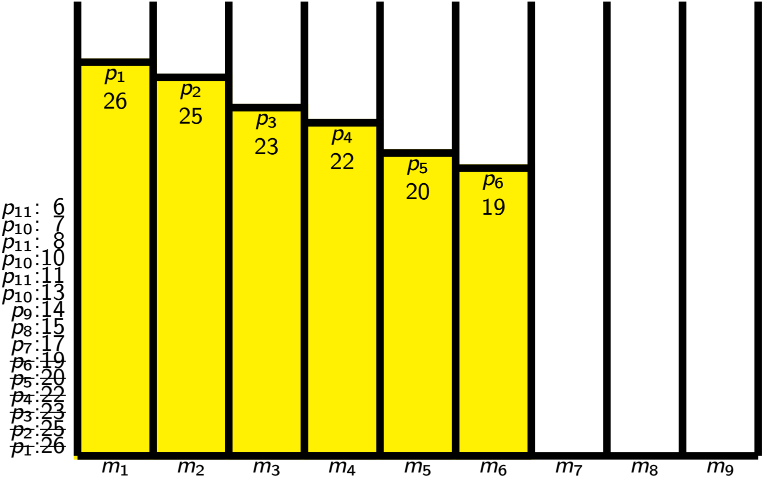
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



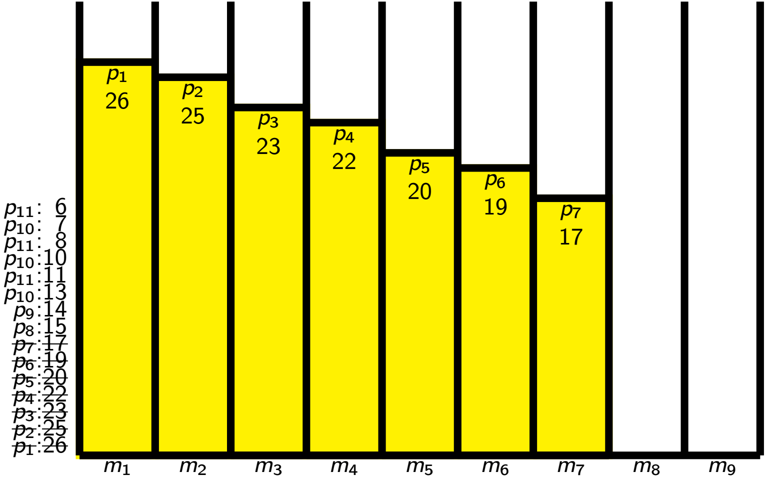
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



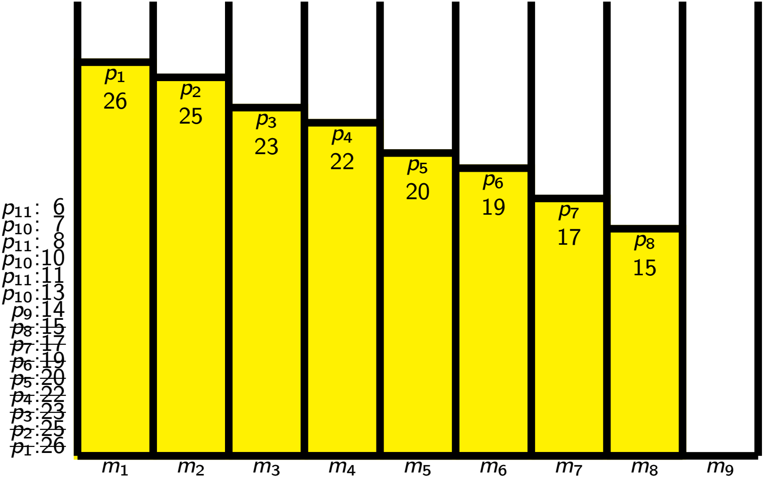
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



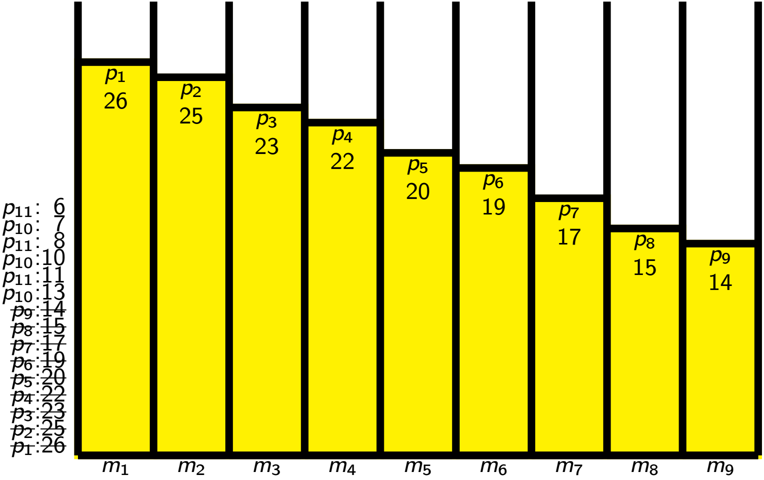
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



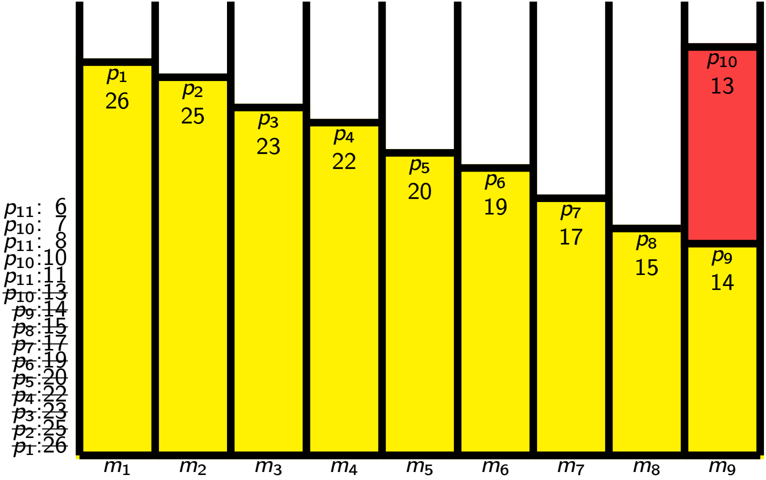
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



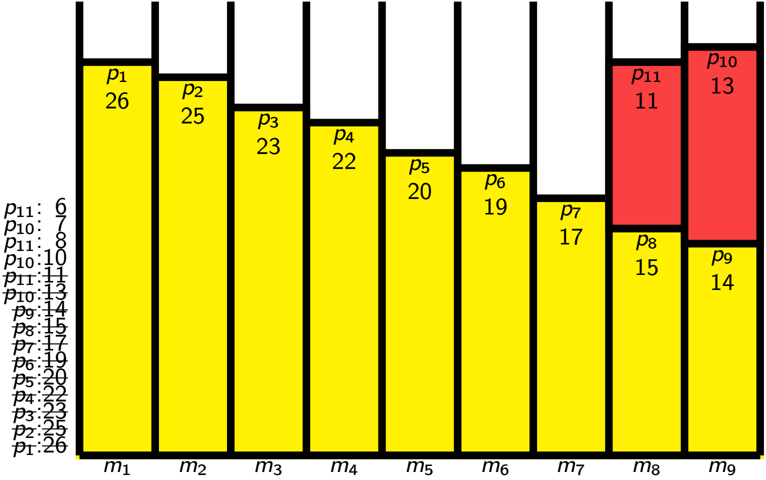
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



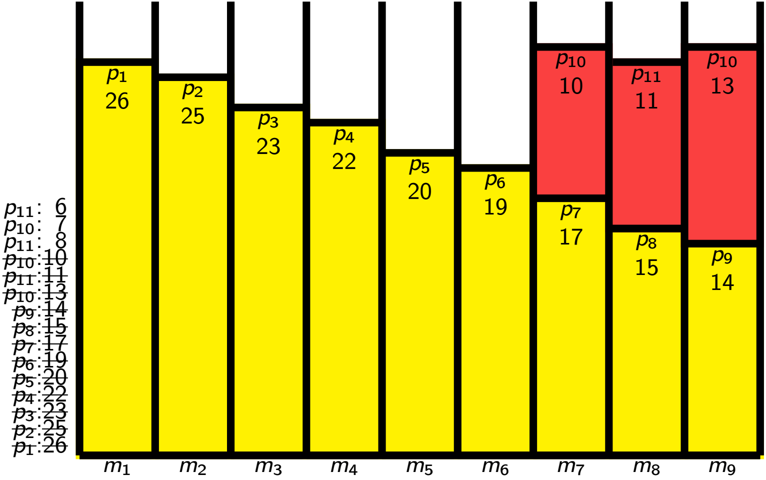
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



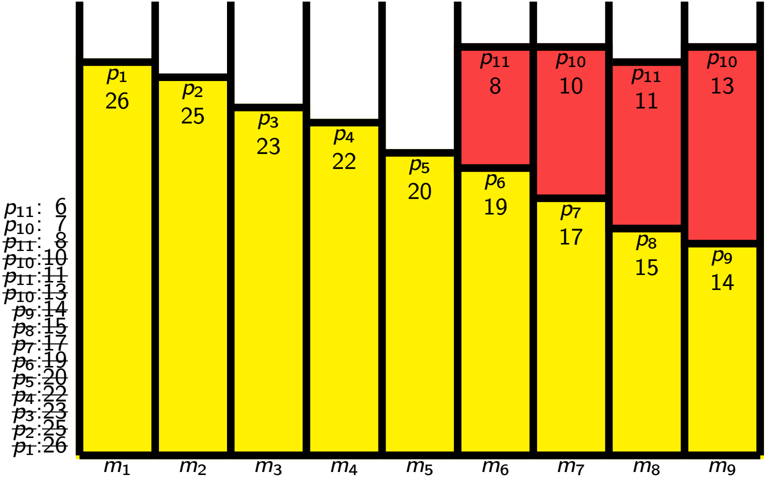
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



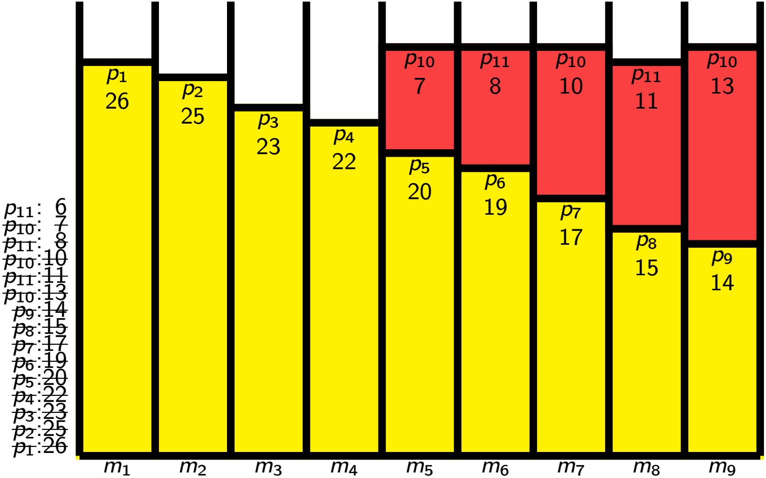
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



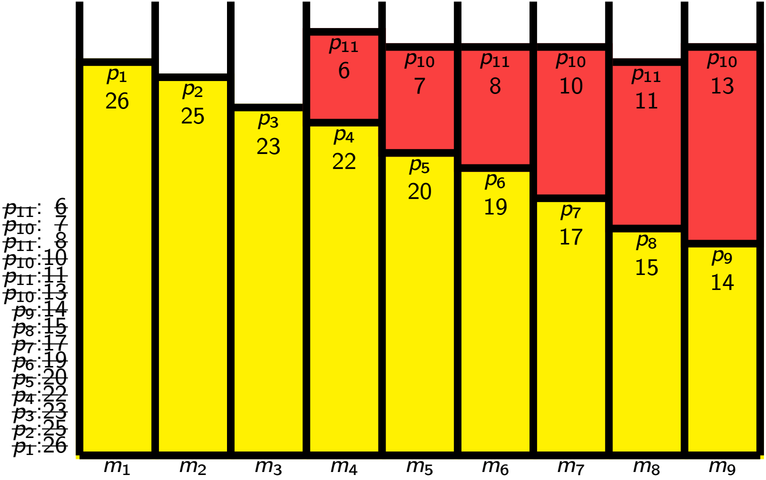
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



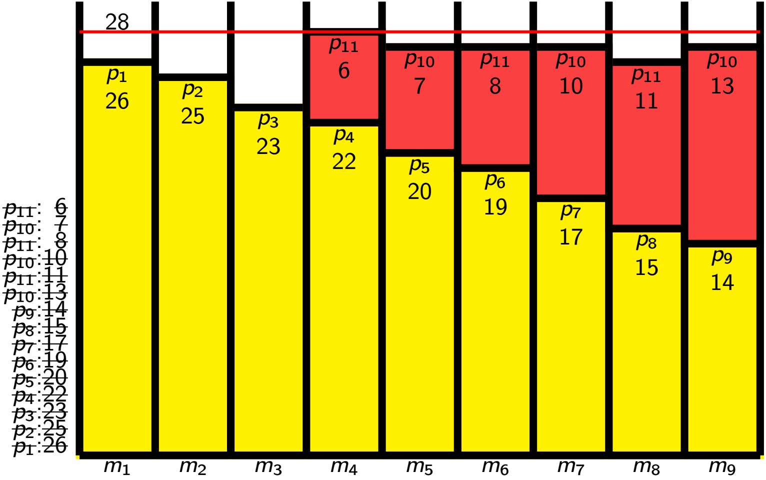
Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



Beispiel zu Heuristik 2 (Nur zwei Jobs pro Maschine)



Approximation Heuristik 2

Theorem (Graham 1969)

Die LPT Heuristik hat einen Approximationsfaktor von 4/3.

Beweis durch Widerspruch:

Approximation Heuristik 2

Theorem (Graham 1969)

Die LPT Heuristik hat einen Approximationsfaktor von $4/3$.

Beweis durch Widerspruch:

- Angenommen, es gibt Eingabeinstanz $p_1, p_2, \ldots, p_n$ auf m Maschinen, mit einem Makespan von $\tau > 4/3 \cdot opt$ und n minimal gewählt.

Approximation Heuristik 2

Theorem (Graham 1969)

Die LPT Heuristik hat einen Approximationsfaktor von 4/3.

Beweis durch Widerspruch:

- Angenommen, es gibt Eingabeinstanz $p_1, p_2, \ldots, p_n$ auf m Maschinen, mit einem Makespan von $\tau > 4/3 \cdot opt$ und n minimal gewählt.
- Seien $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.

Approximation Heuristik 2

Theorem (Graham 1969)

Die LPT Heuristik hat einen Approximationsfaktor von 4/3.

Beweis durch Widerspruch:

- Angenommen, es gibt Eingabeinstanz $p_1, p_2, \ldots, p_n$ auf m Maschinen, mit einem Makespan von $\tau > 4/3 \cdot opt$ und n minimal gewählt.
- Seien $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.
- Da n minimal gewählt wurde, ist n der Job, der als letzter fertig wird.

Approximation Heuristik 2

Theorem (Graham 1969)

Die LPT Heuristik hat einen Approximationsfaktor von 4/3.

Beweis durch Widerspruch:

- Angenommen, es gibt Eingabeinstanz $p_1, p_2, \ldots, p_n$ auf m Maschinen, mit einem Makespan von $\tau > 4/3 \cdot opt$ und n minimal gewählt.
- Seien $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.
- Da n minimal gewählt wurde, ist n der Job, der als letzter fertig wird.
- Der Job n wurde auf die am wenigsten belastete Maschine platziert.

Approximation Heuristik 2

Theorem (Graham 1969)

Die LPT Heuristik hat einen Approximationsfaktor von 4/3.

Beweis durch Widerspruch:

- Angenommen, es gibt Eingabeinstanz $p_1, p_2, \ldots, p_n$ auf m Maschinen, mit einem Makespan von $\tau > 4/3 \cdot opt$ und n minimal gewählt.
- Seien $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.
- Da n minimal gewählt wurde, ist n der Job, der als letzter fertig wird.
- Der Job n wurde auf die am wenigsten belastete Maschine platziert.
- Zu diesem Zeitpunkt war die Last der Maschine höchstens:

$$\frac{1}{m} \cdot \sum_{i=1}^{n-1} p_i \leqslant opt.$$

Approximation Heuristik 2

Theorem (Graham 1969)

Die LPT Heuristik hat einen Approximationsfaktor von $4/3$.

Beweis durch Widerspruch:

- Angenommen, es gibt Eingabeinstanz $p_1, p_2, \ldots, p_n$ auf m Maschinen, mit einem Makespan von $\tau > 4/3 \cdot opt$ und n minimal gewählt.
- Seien $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_n$.
- Da n minimal gewählt wurde, ist n der Job, der als letzter fertig wird.
- Der Job n wurde auf die am wenigsten belastete Maschine platziert.
- Zu diesem Zeitpunkt war die Last der Maschine höchstens:

$$\frac{1}{m} \cdot \sum_{i=1}^{n-1} p_i \leqslant opt.$$

- Damit nun im nächsten Schritt ein Faktor von τ auftritt muss gelten:

$$p_n > 1/3 \cdot opt.$$

Beweis

$p_n > \frac{1}{3} \cdot opt$

- Wegen der Sortierung der Jobs gilt damit: $\forall i : p_i > \frac{1}{3} \cdot opt$.

Beweis

$p_n > \frac{1}{3} \cdot opt$

- Wegen der Sortierung der Jobs gilt damit: $\forall i : p_i > \frac{1}{3} \cdot opt$.
- Also passen höchstens zwei Jobs auf jede Maschine.

Beweis

$p_n > \frac{1}{3} \cdot opt$

- Wegen der Sortierung der Jobs gilt damit: $\forall i : p_i > \frac{1}{3} \cdot opt$.
- Also passen höchstens zwei Jobs auf jede Maschine.
- Es gibt daher auch höchstens $n \leqslant 2 \cdot m$ Jobs.

Beweis

$p_n > \frac{1}{3} \cdot opt$

- Wegen der Sortierung der Jobs gilt damit: $\forall i : p_i > \frac{1}{3} \cdot opt$.
- Also passen höchstens zwei Jobs auf jede Maschine.
- Es gibt daher auch höchstens $n \leqslant 2 \cdot m$ Jobs.
- Damit ist aber folgende Platzierung optimal:

Beweis

$p_n > \frac{1}{3} \cdot opt$

- Wegen der Sortierung der Jobs gilt damit: $\forall i : p_i > \frac{1}{3} \cdot opt$.
- Also passen höchstens zwei Jobs auf jede Maschine.
- Es gibt daher auch höchstens $n \leqslant 2 \cdot m$ Jobs.
- Damit ist aber folgende Platzierung optimal:
- Platziere Job i für $i \leqslant m$ auf Maschine i.

Beweis

$p_n > \frac{1}{3} \cdot opt$

- Wegen der Sortierung der Jobs gilt damit: $\forall i : p_i > \frac{1}{3} \cdot opt$.
- Also passen höchstens zwei Jobs auf jede Maschine.
- Es gibt daher auch höchstens $n \leqslant 2 \cdot m$ Jobs.
- Damit ist aber folgende Platzierung optimal:
- Platziere Job i für $i \leqslant m$ auf Maschine i.
- Platziere Job i für $i > m$ auf Maschine $2 \cdot m - i + 1$.

Beweis

$p_n > \frac{1}{3} \cdot opt$

- Wegen der Sortierung der Jobs gilt damit: $\forall i : p_i > \frac{1}{3} \cdot opt$.
- Also passen höchstens zwei Jobs auf jede Maschine.
- Es gibt daher auch höchstens $n \leqslant 2 \cdot m$ Jobs.
- Damit ist aber folgende Platzierung optimal:
- Platziere Job i für $i \leqslant m$ auf Maschine i.
- Platziere Job i für $i > m$ auf Maschine $2 \cdot m - i + 1$.
- Das ist aber die Platzierung von der LPT Heuristik.

Beweis

$p_n > \frac{1}{3} \cdot opt$

- Wegen der Sortierung der Jobs gilt damit: $\forall i : p_i > \frac{1}{3} \cdot opt$.
- Also passen höchstens zwei Jobs auf jede Maschine.
- Es gibt daher auch höchstens $n \leqslant 2 \cdot m$ Jobs.
- Damit ist aber folgende Platzierung optimal:
- Platziere Job i für $i \leqslant m$ auf Maschine i.
- Platziere Job i für $i > m$ auf Maschine $2 \cdot m - i + 1$.
- Das ist aber die Platzierung von der LPT Heuristik.
- Widerspruch.

Motivation und Definition

- Im nächsten Abschnitt soll das Scheduling auf identischen Maschinen mit einem beliebigen konstanten Faktor approximiert werden.

Motivation und Definition

- Im nächsten Abschnitt soll das Scheduling auf identischen Maschinen mit einem beliebigen konstanten Faktor approximiert werden.
- Dabei wird die Lösung des folgenden Problems hilfreich sein.

Motivation und Definition

- Im nächsten Abschnitt soll das Scheduling auf identischen Maschinen mit einem beliebigen konstanten Faktor approximiert werden.
- Dabei wird die Lösung des folgenden Problems hilfreich sein.
- Bemerkung vorweg: Bin Packing entspricht einem Scheduling, bei der es eine Schranke b für den Makespan gibt.

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.
  - $w_1, w_2, \ldots, w_n$ mit $\{w_1, w_2, \ldots, w_n\} \subseteq \{1, 2, \ldots, k\}$.

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.
  - $w_1, w_2, \ldots, w_n$ mit $\{w_1, w_2, \ldots, w_n\} \subseteq \{1, 2, \ldots, k\}$.
  - Zwei Zahlen $m, b \in \mathbb{N}$ mit: $m \geqslant 1$ und $b \geqslant k$.

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.
  - $w_1, w_2, \ldots, w_n$ mit $\{w_1, w_2, \ldots, w_n\} \subseteq \{1, 2, \ldots, k\}$.
  - Zwei Zahlen $m, b \in \mathbb{N}$ mit: $m \geqslant 1$ und $b \geqslant k$.
- Gesucht:

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.
  - $w_1, w_2, \ldots, w_n$ mit $\{w_1, w_2, \ldots, w_n\} \subseteq \{1, 2, \ldots, k\}$.
  - Zwei Zahlen $m, b \in \mathbb{N}$ mit: $m \geqslant 1$ und $b \geqslant k$.
- Gesucht:
  - $z : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$ mit:

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.
  - $w_1, w_2, \ldots, w_n$ mit $\{w_1, w_2, \ldots, w_n\} \subseteq \{1, 2, \ldots, k\}$.
  - Zwei Zahlen $m, b \in \mathbb{N}$ mit: $m \geqslant 1$ und $b \geqslant k$.
- Gesucht:
  - $z : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$ mit:
  - $\forall i \in \{1, 2, \ldots, m\} : \sum_{j \in \{1,2,\ldots,n\} : z(j)=i} w_j \leqslant b$.

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.
  - $w_1, w_2, \ldots, w_n$ mit $\{w_1, w_2, \ldots, w_n\} \subseteq \{1, 2, \ldots, k\}$.
  - Zwei Zahlen $m, b \in \mathbb{N}$ mit: $m \geqslant 1$ und $b \geqslant k$.
- Gesucht:
  - $z : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$ mit:
  - $\forall i \in \{1, 2, \ldots, m\} : \sum_{j \in \{1,2,\ldots,n\} : z(j) = i} w_j \leqslant b$.

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.
  - $w_1, w_2, \ldots, w_n$ mit $\{w_1, w_2, \ldots, w_n\} \subseteq \{1, 2, \ldots, k\}$.
  - Zwei Zahlen $m, b \in \mathbb{N}$ mit: $m \geqslant 1$ und $b \geqslant k$.
- Gesucht:
  - $z : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$ mit:
  - $\forall i \in \{1, 2, \ldots, m\} : \sum_{j \in \{1,2,\ldots,n\}: z(j)=i} w_j \leqslant b$.

- Die w_i sind die Gewichte der Objekte.

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.
  - $w_1, w_2, \ldots, w_n$ mit $\{w_1, w_2, \ldots, w_n\} \subseteq \{1, 2, \ldots, k\}$.
  - Zwei Zahlen $m, b \in \mathbb{N}$ mit: $m \geqslant 1$ und $b \geqslant k$.
- Gesucht:
  - $z : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$ mit:
  - $\forall i \in \{1, 2, \ldots, m\} : \sum_{j \in \{1,2,\ldots,n\}: z(j)=i} w_j \leqslant b$.

- Die w_i sind die Gewichte der Objekte.
- Die Funktion z ist die Verteilung der Objekte auf m Bins.

Motivation und Definition

Definition (Bin Packing mit eingeschränkten Gewichten)

- Gegeben:
  - n Objekte: $\{1, 2, \ldots, n\}$.
  - $w_1, w_2, \ldots, w_n$ mit $\{w_1, w_2, \ldots, w_n\} \subseteq \{1, 2, \ldots, k\}$.
  - Zwei Zahlen $m, b \in \mathbb{N}$ mit: $m \geqslant 1$ und $b \geqslant k$.
- Gesucht:
  - $z : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$ mit:
  - $\forall i \in \{1, 2, \ldots, m\} : \sum_{j \in \{1,2,\ldots,n\}: z(j)=i} w_j \leqslant b$.

- Die w_i sind die Gewichte der Objekte.
- Die Funktion z ist die Verteilung der Objekte auf m Bins.
- Jeder Bin kann maximal b aufnehmen.

Aussage

Theorem

Das Bin Packing Problem mit eingeschränkten Gewichten kann in Zeit $O((n+1)^k \cdot (b+1)^k / k!)$ *gelöst werden.*

Aussage

Theorem

Das Bin Packing Problem mit eingeschränkten Gewichten kann in Zeit $O((n+1)^k \cdot (b+1)^k/k!)$ gelöst werden.

Aussage

Theorem

Das Bin Packing Problem mit eingeschränkten Gewichten kann in Zeit $O((n+1)^k \cdot (b+1)^k/k!)$ gelöst werden.

- Die k möglichen Gewichte schränken das Problem ein.

Aussage

Theorem

Das Bin Packing Problem mit eingeschränkten Gewichten kann in Zeit $O((n+1)^k \cdot (b+1)^k/k!)$ gelöst werden.

- Die k möglichen Gewichte schränken das Problem ein.
- Zwei Objekte mit dem gleichen Gewicht sind austauschbar.

Aussage

Theorem

Das Bin Packing Problem mit eingeschränkten Gewichten kann in Zeit $O((n+1)^k \cdot (b+1)^k / k!)$ gelöst werden.

- Die k möglichen Gewichte schränken das Problem ein.
- Zwei Objekte mit dem gleichen Gewicht sind austauschbar.
- Daher untersuche nur Lösungen nach der Anzahl der Objekte vom gleichen Gewicht.

Aussage

Theorem

Das Bin Packing Problem mit eingeschränkten Gewichten kann in Zeit $O((n+1)^k \cdot (b+1)^k / k!)$ gelöst werden.

- Die k möglichen Gewichte schränken das Problem ein.
- Zwei Objekte mit dem gleichen Gewicht sind austauschbar.
- Daher untersuche nur Lösungen nach der Anzahl der Objekte vom gleichen Gewicht.
- Nutze dynamische Programmierung.

Aussage

Theorem

Das Bin Packing Problem mit eingeschränkten Gewichten kann in Zeit $O((n+1)^k \cdot (b+1)^k/k!)$ gelöst werden.

- Die k möglichen Gewichte schränken das Problem ein.
- Zwei Objekte mit dem gleichen Gewicht sind austauschbar.
- Daher untersuche nur Lösungen nach der Anzahl der Objekte vom gleichen Gewicht.
- Nutze dynamische Programmierung.
- Untersuche die Anzahl der notwendigen Bins für n_i Objekte mit Gewicht w_i $(1 \leqslant i \leqslant k)$.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Sei $f(n_1, n_2, \ldots, n_k)$ die minimale Anzahl von Bins der Größe b, die n_i Objekte mit Gewicht w_i $(1 \leqslant i \leqslant k)$ aufnehmen können.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Sei $f(n_1, n_2, \ldots, n_k)$ die minimale Anzahl von Bins der Größe b, die n_i Objekte mit Gewicht w_i $(1 \leqslant i \leqslant k)$ aufnehmen können.
- Setze $Q = \{(q_1, q_2, \ldots, q_k) \mid f(q_1, q_2, \ldots, q_k) = 1\}$, d.h. die Gewichtskombinationen, die in einen Bin passen.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Sei $f(n_1, n_2, \ldots, n_k)$ die minimale Anzahl von Bins der Größe b, die n_i Objekte mit Gewicht w_i $(1 \leqslant i \leqslant k)$ aufnehmen können.
- Setze $Q = \{(q_1, q_2, \ldots, q_k) \mid f(q_1, q_2, \ldots, q_k) = 1\}$, d.h. die Gewichtskombinationen, die in einen Bin passen.
- Damit kann f rekursiv beschrieben werden:

$$f(n_1, n_2, \ldots, n_k) = 1 + \min_{(q_\mathbf{1}, q_\mathbf{2}, \ldots, q_k) \in Q} f(n_1 - q_1, n_2 - q_2, \ldots, n_k - q_k).$$

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Sei $f(n_1, n_2, \ldots, n_k)$ die minimale Anzahl von Bins der Größe b, die n_i Objekte mit Gewicht w_i $(1 \leqslant i \leqslant k)$ aufnehmen können.
- Setze $Q = \{(q_1, q_2, \ldots, q_k) \mid f(q_1, q_2, \ldots, q_k) = 1\}$, d.h. die Gewichtskombinationen, die in einen Bin passen.
- Damit kann f rekursiv beschrieben werden:

$$f(n_1, n_2, \ldots, n_k) = 1 + \min_{(q_1, q_2, \ldots, q_k) \in Q} f(n_1 - q_1, n_2 - q_2, \ldots, n_k - q_k).$$

- Im Weiteren wird gezeigt, wie diese Werte bestimmt werden.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Sei $f(n_1, n_2, \ldots, n_k)$ die minimale Anzahl von Bins der Größe b, die n_i Objekte mit Gewicht w_i $(1 \leqslant i \leqslant k)$ aufnehmen können.
- Setze $Q = \{(q_1, q_2, \ldots, q_k) \mid f(q_1, q_2, \ldots, q_k) = 1\}$, d.h. die Gewichtskombinationen, die in einen Bin passen.
- Damit kann f rekursiv beschrieben werden:

$$f(n_1, n_2, \ldots, n_k) = 1 + \min_{(q_1, q_2, \ldots, q_k) \in Q} f(n_1 - q_1, n_2 - q_2, \ldots, n_k - q_k).$$

- Im Weiteren wird gezeigt, wie diese Werte bestimmt werden.
- Sei vorher: $c_i = |\{j \mid j \in \{1, 2, \ldots, n\} \wedge w(j) = i\}|$.
  D.h. c_i ist die Anzahl der Objekte mit Gewicht i.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Sei $f(n_1, n_2, \ldots, n_k)$ die minimale Anzahl von Bins der Größe b, die n_i Objekte mit Gewicht w_i $(1 \leqslant i \leqslant k)$ aufnehmen können.
- Setze $Q = \{(q_1, q_2, \ldots, q_k) \mid f(q_1, q_2, \ldots, q_k) = 1\}$, d.h. die Gewichtskombinationen, die in einen Bin passen.
- Damit kann f rekursiv beschrieben werden:

$$f(n_1, n_2, \ldots, n_k) = 1 + \min_{(q_1, q_2, \ldots, q_k) \in Q} f(n_1 - q_1, n_2 - q_2, \ldots, n_k - q_k).$$

- Im Weiteren wird gezeigt, wie diese Werte bestimmt werden.
- Sei vorher: $c_i = |\{j \mid j \in \{1, 2, \ldots, n\} \wedge w(j) = i\}|$.
  D.h. c_i ist die Anzahl der Objekte mit Gewicht i.
- Das Bin Packing Problem mit eingeschränkten Gewichten hat eine Lösung genau dann, wenn:

$$f(c_1, c_2, \ldots, c_k) \leqslant m.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3$, $b = 8$, $p_1 = 3$, $p_2 = 3$, $p_3 = 4$, $p_4 = 4$, $p_5 = 5$, $p_6 = 5$.

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- D.h. $c_1 = 0, c_2 = 0, c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3$, $b = 8$, $p_1 = 3$, $p_2 = 3$, $p_3 = 4$, $p_4 = 4$, $p_5 = 5$, $p_6 = 5$.
- D.h. $c_1 = 0, c_2 = 0, c_3 = 2$, $c_4 = 2$, $c_5 = 2$.

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- D.h. $c_1 = 0, c_2 = 0, c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

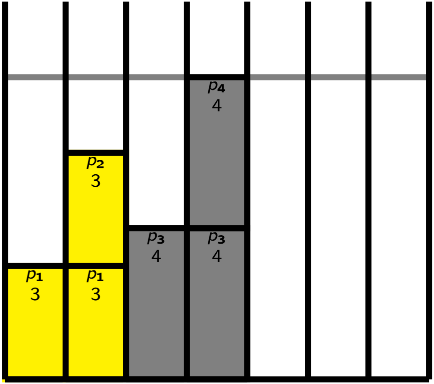
$$\begin{aligned} Q' \quad = \quad \{ \quad & (0,0,1,0,0), \\ & (0,0,2,0,0), \\ & (0,0,0,1,0), \end{aligned}$$

Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- D.h. $c_1 = 0, c_2 = 0, c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

$$
\begin{aligned}
Q' \quad = \quad \{ \quad & (0,0,1,0,0), \\
& (0,0,2,0,0), \\
& (0,0,0,1,0), \\
& (0,0,0,2,0),
\end{aligned}
$$

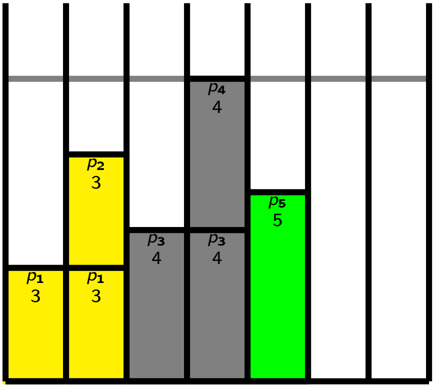


Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- D.h. $c_1 = 0, c_2 = 0, c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

$$\begin{aligned} Q' \quad = \quad \{ \quad & (0,0,1,0,0), \\ & (0,0,2,0,0), \\ & (0,0,0,1,0), \\ & (0,0,0,2,0), \\ & (0,0,0,0,1), \end{aligned}$$

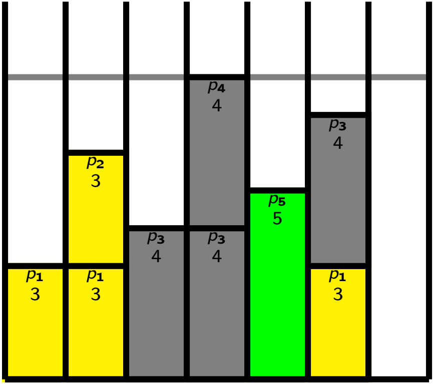


Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- D.h. $c_1 = 0, c_2 = 0, c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

$$Q' = \{ (0,0,1,0,0), (0,0,2,0,0), (0,0,0,1,0), (0,0,0,2,0), (0,0,0,0,1), (0,0,1,1,0),$$

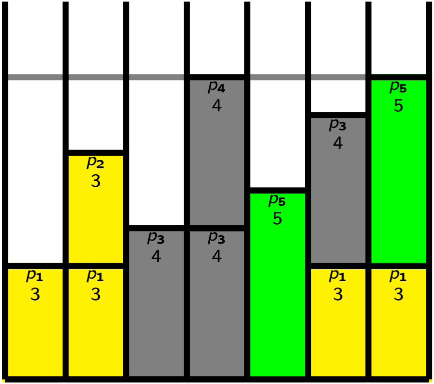


Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- D.h. $c_1 = 0, c_2 = 0, c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

$$Q' = \{ (0,0,1,0,0), (0,0,2,0,0), (0,0,0,1,0), (0,0,0,2,0), (0,0,0,0,1), (0,0,1,1,0), (0,0,1,0,1) \}$$



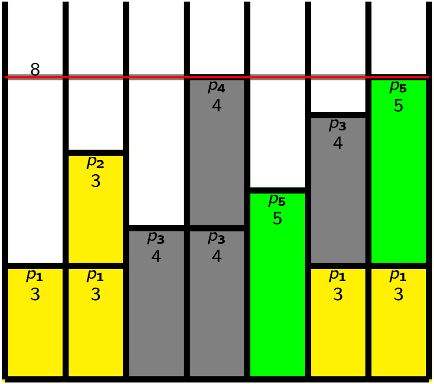
Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- D.h. $c_1 = 0, c_2 = 0, c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

$$
\begin{aligned}
Q' = \{ \quad & (0,0,1,0,0), \\
& (0,0,2,0,0), \\
& (0,0,0,1,0), \\
& (0,0,0,2,0), \\
& (0,0,0,0,1), \\
& (0,0,1,1,0), \\
& (0,0,1,0,1) \quad \}
\end{aligned}
$$

$$
\begin{aligned}
Q = \{ \quad & (1,0,0), (2,0,0), \\
& (0,1,0), (0,2,0), \\
& (0,0,1), (1,1,0), \\
& (1,0,1) \quad \}
\end{aligned}
$$



Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

$$f(2,2,2) = 1 + \min \left\{ \phantom{\begin{matrix} \\ \\ \\ \\ \end{matrix}} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k/k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \left\{ \begin{array}{l} f(1,2,2) \\ \\ \\ \\ \\ \\ \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) \\ f(0,2,2) \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) \\ f(0,2,2) \\ f(2,1,2) \\ \\ \\ \\ \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) \\ f(0,2,2) \\ f(2,1,2) \\ f(2,0,2) \\ \\ \\ \\ \end{cases}$$

Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) \\ f(0,2,2) \\ f(2,1,2) \\ f(2,0,2) \\ f(2,2,1) \\ \\ \\ \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) \\ f(0,2,2) \\ f(2,1,2) \\ f(2,0,2) \\ f(2,2,1) \\ f(1,1,2) \\ \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) \\ f(0,2,2) \\ f(2,1,2) \\ f(2,0,2) \\ f(2,2,1) \\ f(1,1,2) \\ f(1,2,1) \end{cases}$$

Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \left\{ \begin{array}{l} f(1,2,2) \\ f(0,2,2) \\ f(2,1,2) \\ f(2,0,2) \\ f(2,2,1) \\ f(1,1,2) \\ f(1,2,1) \end{array} \right.$$

$$\lceil (1 \cdot 3 + 2 \cdot 4 + 2 \cdot 5)/8 \rceil = 3$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \\ f(2,0,2) & \\ f(2,2,1) & \\ f(1,1,2) & \\ f(1,2,1) & \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \left\{ \begin{array}{ll} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \\ f(2,2,1) & \\ f(1,1,2) & \\ f(1,2,1) & \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2$.
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}$.

$$f(2,2,2) = 1 + \min \left\{ \begin{array}{ll} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \lceil (2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8 \rceil = 2 \\ f(2,2,1) & \\ f(1,1,2) & \\ f(1,2,1) & \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \lceil (2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8 \rceil = 2 \\ f(2,2,1) & \lceil (2\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 3 \\ f(1,1,2) & \\ f(1,2,1) & \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$

$$f(2,2,2) = 1 + \min \left\{ \begin{array}{ll} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \lceil (2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8 \rceil = 2 \\ f(2,2,1) & \lceil (2\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 3 \\ f(1,1,2) & \lceil (1\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(1,2,1) & \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \left\{ \begin{array}{ll} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \lceil (2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8 \rceil = 2 \\ f(2,2,1) & \lceil (2\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 3 \\ f(1,1,2) & \lceil (1\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(1,2,1) & \lceil (1\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 2 \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) & \lceil(1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8\rceil = 3 \\ f(0,2,2) & \lceil(0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8\rceil = 3 \\ f(2,1,2) & \lceil(2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8\rceil = 3 \\ f(2,0,2) & \lceil(2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8\rceil = 2 \\ f(2,2,1) = 3 & \lceil(2\cdot 3 + 2\cdot 4 + 1\cdot 5)/8\rceil = 3 \\ f(1,1,2) & \lceil(1\cdot 3 + 1\cdot 4 + 2\cdot 5)/8\rceil = 3 \\ f(1,2,1) & \lceil(1\cdot 3 + 2\cdot 4 + 1\cdot 5)/8\rceil = 2 \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \lceil (2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8 \rceil = 2 \\ f(2,2,1) = 3 & \lceil (2\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 3 \\ f(1,1,2) & \lceil (1\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(1,2,1) & \lceil (1\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 2 \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \left\{ \begin{array}{ll} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \lceil (2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8 \rceil = 2 \\ f(2,2,1) = 3 & \lceil (2\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 3 \\ f(1,1,2) & \lceil (1\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(1,2,1) & \lceil (1\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 2 \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \lceil (2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8 \rceil = 2 \\ f(2,2,1) = 3 & \lceil (2\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 3 \\ f(1,1,2) & \lceil (1\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(1,2,1) & \lceil (1\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 2 \end{cases}$$

- Damit gilt $f(2,2,2) \leqslant 4$.

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k/k!)$

- Eingabe: $m=3,\ b=8,\ c_3=2,\ c_4=2,\ c_5=2.$
- $Q=\{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$

$$f(2,2,2) = 1+\min\begin{cases} f(1,2,2) & \lceil(1\cdot 3+2\cdot 4+2\cdot 5)/8\rceil = 3\\ f(0,2,2) & \lceil(0\cdot 3+2\cdot 4+2\cdot 5)/8\rceil = 3\\ f(2,1,2) & \lceil(2\cdot 3+1\cdot 4+2\cdot 5)/8\rceil = 3\\ f(2,0,2) & \lceil(2\cdot 3+0\cdot 4+2\cdot 5)/8\rceil = 2\\ f(2,2,1)=3 & \lceil(2\cdot 3+2\cdot 4+1\cdot 5)/8\rceil = 3\\ f(1,1,2) & \lceil(1\cdot 3+1\cdot 4+2\cdot 5)/8\rceil = 3\\ f(1,2,1) & \lceil(1\cdot 3+2\cdot 4+1\cdot 5)/8\rceil = 2 \end{cases}$$

- Damit gilt $f(2,2,2)\leqslant 4$.
- Und weiter $3\leqslant f(2,2,2)\leqslant 4$.

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2$.
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}$.

$$f(2,2,2) = 1 + \min \begin{cases} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \lceil (2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8 \rceil = 2 \\ f(2,2,1) = 3 & \lceil (2\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 3 \\ f(1,1,2) & \lceil (1\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(1,2,1) & \lceil (1\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 2 \end{cases}$$

- Damit gilt $f(2,2,2) \leqslant 4$.
- Und weiter $3 \leqslant f(2,2,2) \leqslant 4$.
- Untersuche nun ob $f(2,0,2) = 2$ gilt.

Beispiel

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2$.
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}$.

$$f(2,2,2) = 1 + \min \left\{ \begin{array}{ll} f(1,2,2) & \lceil (1\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(0,2,2) & \lceil (0\cdot 3 + 2\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,1,2) & \lceil (2\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(2,0,2) & \lceil (2\cdot 3 + 0\cdot 4 + 2\cdot 5)/8 \rceil = 2 \\ f(2,2,1) = 3 & \lceil (2\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 3 \\ f(1,1,2) & \lceil (1\cdot 3 + 1\cdot 4 + 2\cdot 5)/8 \rceil = 3 \\ f(1,2,1) & \lceil (1\cdot 3 + 2\cdot 4 + 1\cdot 5)/8 \rceil = 2 \end{array} \right.$$

- Damit gilt $f(2,2,2) \leqslant 4$.
- Und weiter $3 \leqslant f(2,2,2) \leqslant 4$.
- Untersuche nun ob $f(2,0,2) = 2$ gilt.
- Nur wenn $f(2,0,2) \geqslant 3$ gilt, dann untersuche auch $f(1,2,1)$.

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

$$f(2,0,2) = 1 + \min \left\{ \vphantom{\begin{array}{c}\\ \\ \\ \\ \\ \\ \end{array}} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) \\ \\ \\ \\ \\ \end{cases}$$

Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2$.
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}$.
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \left\{ \begin{array}{l} f(1,0,2) \\ f(0,0,2) \\ \\ \\ \\ \\ \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2$.
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}$.
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) \\ f(0,0,2) \\ f(2,0,2) \\ \\ \\ \\ \\ \end{cases}$$

Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \left\{ \begin{array}{l} f(1,0,2) \\ f(0,0,2) \\ f(2,0,2) \\ f(2,0,2) \\ \\ \\ \\ \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k/k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) \\ f(0,0,2) \\ f(2,0,2) \\ f(2,0,2) \\ f(2,0,1) \\ \\ \\ \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) \\ f(0,0,2) \\ f(2,0,2) \\ f(2,0,2) \\ f(2,0,1) \\ f(1,0,2) \\ \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2$.
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}$.
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) \\ f(0,0,2) \\ f(2,0,2) \\ f(2,0,2) \\ f(2,0,1) \\ f(1,0,2) \\ f(1,0,1) \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) \\ f(0,0,2) \\ f(2,0,2) \\ f(2,0,2) \\ f(2,0,1) \\ f(1,0,2) \\ f(1,0,1) \end{cases} \qquad \lceil (1 \cdot 3 + 2 \cdot 5)/8 \rceil = 2$$

Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2$.
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}$.
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \\ f(2,0,2) & \\ f(2,0,1) & \\ f(1,0,2) & \\ f(1,0,1) & \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \left\{ \begin{array}{ll} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \\ f(2,0,1) & \\ f(1,0,2) & \\ f(1,0,1) & \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,1) & \\ f(1,0,2) & \\ f(1,0,1) & \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \left\{ \begin{array}{ll} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,1) & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 2 \\ f(1,0,2) & \\ f(1,0,1) & \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,1) & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 2 \\ f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(1,0,1) & \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) & \lceil (1 \cdot 3 + 2 \cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0 \cdot 3 + 2 \cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,1) & \lceil (1 \cdot 3 + 1 \cdot 5)/8 \rceil = 2 \\ f(1,0,2) & \lceil (1 \cdot 3 + 2 \cdot 5)/8 \rceil = 2 \\ f(1,0,1) & \lceil (1 \cdot 3 + 1 \cdot 5)/8 \rceil = 1 \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,1) & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 2 \\ f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(1,0,1) = 1 & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 1 \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \left\{ \begin{array}{ll} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,1) & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 2 \\ f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(1,0,1) = 1 & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 1 \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,1) & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 2 \\ f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(1,0,1) = 1 & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 1 \end{cases}$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \left\{ \begin{array}{ll} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,1) & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 2 \\ f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(1,0,1) = 1 & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 1 \end{array} \right.$$

Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$

$$f(2,0,2) = 1 + \min \begin{cases} f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(0,0,2) & \lceil (0\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,2) & \text{Keine Lösung} \\ f(2,0,1) & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 2 \\ f(1,0,2) & \lceil (1\cdot 3 + 2\cdot 5)/8 \rceil = 2 \\ f(1,0,1) = 1 & \lceil (1\cdot 3 + 1\cdot 5)/8 \rceil = 1 \end{cases}$$

- Damit gilt $f(2,0,2) = 2.$

Beispiel

$O((n+1)^k \cdot (b+1)^k/k!)$

- Eingabe: $m=3,\ b=8,\ c_3=2,\ c_4=2,\ c_5=2.$
- $Q=\{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2)=1+\min\{3, f(2,0,2), f(1,2,1)\}$

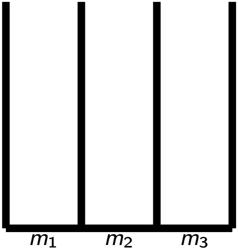
$$f(2,0,2)=1+\min\left\{\begin{array}{ll} f(1,0,2) & \lceil(1\cdot 3+2\cdot 5)/8\rceil=2\\ f(0,0,2) & \lceil(0\cdot 3+2\cdot 5)/8\rceil=2\\ f(2,0,2) & \text{Keine Lösung}\\ f(2,0,2) & \text{Keine Lösung}\\ f(2,0,1) & \lceil(1\cdot 3+1\cdot 5)/8\rceil=2\\ f(1,0,2) & \lceil(1\cdot 3+2\cdot 5)/8\rceil=2\\ f(1,0,1)=1 & \lceil(1\cdot 3+1\cdot 5)/8\rceil=1 \end{array}\right.$$

- Damit gilt $f(2,0,2)=2$.
- Und damit auch $f(2,2,2)=3$.

Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$

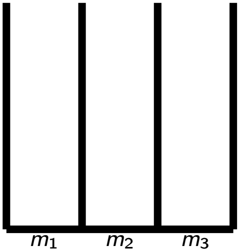


Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$

- $f(1, 0, 1) = 1$

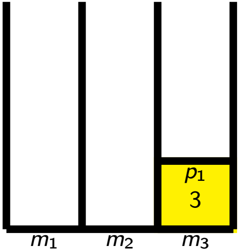


Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Eingabe: $m = 3$, $b = 8$, $c_3 = 2$, $c_4 = 2$, $c_5 = 2$.
- $p_1 = 3$, $p_2 = 3$, $p_3 = 4$, $p_4 = 4$, $p_5 = 5$, $p_6 = 5$.
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}$.

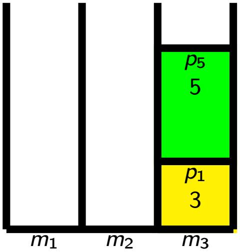
- $f(1, 0, 1) = 1$



Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

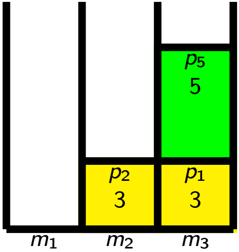
- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- $Q = \{(1,0,0), (2,0,0), (0,1,0), (0,2,0), (0,0,1), (1,1,0), (1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$
- $f(1,0,1) = 1$



Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

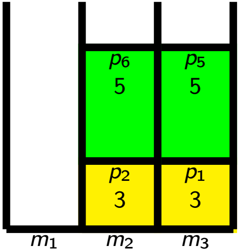
- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$
- $f(2,0,2) = 1 + \min\{2, f(1,0,1)\}$
- $f(1,0,1) = 1$



Beispiel

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

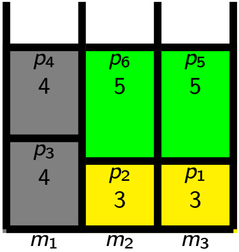
- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$
- $f(2,0,2) = 1 + \min\{2, f(1,0,1)\}$
- $f(1,0,1) = 1$



Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

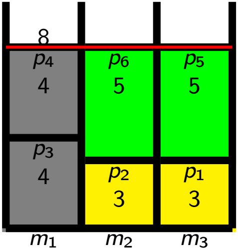
- Eingabe: $m = 3$, $b = 8$, $c_3 = 2$, $c_4 = 2$, $c_5 = 2$.
- $p_1 = 3$, $p_2 = 3$, $p_3 = 4$, $p_4 = 4$, $p_5 = 5$, $p_6 = 5$.
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}$.
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$
- $f(2,0,2) = 1 + \min\{2, f(1,0,1)\}$
- $f(1,0,1) = 1$



Beispiel

$O((n+1)^k \cdot (b+1)^k / k!)$

- Eingabe: $m = 3,\ b = 8,\ c_3 = 2,\ c_4 = 2,\ c_5 = 2.$
- $p_1 = 3,\ p_2 = 3,\ p_3 = 4,\ p_4 = 4,\ p_5 = 5,\ p_6 = 5.$
- $Q = \{(1,0,0),(2,0,0),(0,1,0),(0,2,0),(0,0,1),(1,1,0),(1,0,1)\}.$
- $f(2,2,2) = 1 + \min\{3, f(2,0,2), f(1,2,1)\}$
- $f(2,0,2) = 1 + \min\{2, f(1,0,1)\}$
- $f(1,0,1) = 1$



Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Parameter für f sind alle Tupel aus $\{0, 1, \ldots, n\}^k$.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Parameter für f sind alle Tupel aus $\{0, 1, \ldots, n\}^k$.
- Speichere Werte für f in einer Tabelle der Größe $(n+1)^k$.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Parameter für f sind alle Tupel aus $\{0, 1, \ldots, n\}^k$.
- Speichere Werte für f in einer Tabelle der Größe $(n+1)^k$.
- Löse mittels Dynamischer Programmierung.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Parameter für f sind alle Tupel aus $\{0, 1, \ldots, n\}^k$.
- Speichere Werte für f in einer Tabelle der Größe $(n+1)^k$.
- Löse mittels Dynamischer Programmierung.
- Laufzeitanalyse:

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Parameter für f sind alle Tupel aus $\{0, 1, \ldots, n\}^k$.
- Speichere Werte für f in einer Tabelle der Größe $(n+1)^k$.
- Löse mittels Dynamischer Programmierung.
- Laufzeitanalyse:
  - Es sind $(n+1)^k$ viele Tabelleneinträge zu berechnen.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Parameter für f sind alle Tupel aus $\{0, 1, \ldots, n\}^k$.
- Speichere Werte für f in einer Tabelle der Größe $(n+1)^k$.
- Löse mittels Dynamischer Programmierung.
- Laufzeitanalyse:
  - Es sind $(n+1)^k$ viele Tabelleneinträge zu berechnen.
  - Jeder Tabelleneintrag kostet Zeit $O(|Q|)$.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Parameter für f sind alle Tupel aus $\{0, 1, \ldots, n\}^k$.
- Speichere Werte für f in einer Tabelle der Größe $(n+1)^k$.
- Löse mittels Dynamischer Programmierung.
- Laufzeitanalyse:
  - Es sind $(n+1)^k$ viele Tabelleneinträge zu berechnen.
  - Jeder Tabelleneintrag kostet Zeit $O(|Q|)$.
  - Falls $(q_1, q_2, \ldots, q_k) \in Q$, dann gilt $q_i \in \{0, 1, \ldots, \lfloor b/i \rfloor\}$ für $i \in \{1, 2, \ldots k\}$.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Parameter für f sind alle Tupel aus $\{0, 1, \ldots, n\}^k$.
- Speichere Werte für f in einer Tabelle der Größe $(n+1)^k$.
- Löse mittels Dynamischer Programmierung.
- Laufzeitanalyse:
  - Es sind $(n+1)^k$ viele Tabelleneinträge zu berechnen.
  - Jeder Tabelleneintrag kostet Zeit $O(|Q|)$.
  - Falls $(q_1, q_2, \ldots, q_k) \in Q$, dann gilt $q_i \in \{0, 1, \ldots, \lfloor b/i \rfloor\}$ für $i \in \{1, 2, \ldots k\}$.
  - Damit passen auch nicht mehr als $\lfloor b/i \rfloor$ Objekte in dieselbe Box.

Dynamische Programmierung

$O((n+\mathbf{1})^k \cdot (b+\mathbf{1})^k / k!)$

- Parameter für f sind alle Tupel aus $\{0, 1, \ldots, n\}^k$.
- Speichere Werte für f in einer Tabelle der Größe $(n+1)^k$.
- Löse mittels Dynamischer Programmierung.
- Laufzeitanalyse:
  - Es sind $(n+1)^k$ viele Tabelleneinträge zu berechnen.
  - Jeder Tabelleneintrag kostet Zeit $O(|Q|)$.
  - Falls $(q_1, q_2, \ldots, q_k) \in Q$, dann gilt $q_i \in \{0, 1, \ldots, \lfloor b/i \rfloor\}$ für $i \in \{1, 2, \ldots k\}$.
  - Damit passen auch nicht mehr als $\lfloor b/i \rfloor$ Objekte in dieselbe Box.
  - Daraus folgt: $|Q| \leqslant (b+1)^k / k!$.

Motivation

- Bisher zwei gute einfache Heuristiken

Motivation

- Bisher zwei gute einfache Heuristiken
- Frage: wie gut können wir das Scheduling Problem auf identischen Maschinen approximieren?

Motivation

- Bisher zwei gute einfache Heuristiken
- Frage: wie gut können wir das Scheduling Problem auf identischen Maschinen approximieren?
- Frage: gibt es da ggf. eine untere Schranke, oder kommen wir beliebig nah an das Optimum heran?

Motivation

- Bisher zwei gute einfache Heuristiken
- Frage: wie gut können wir das Scheduling Problem auf identischen Maschinen approximieren?
- Frage: gibt es da ggf. eine untere Schranke, oder kommen wir beliebig nah an das Optimum heran?
- Antwort: wir kommen beliebig nah an das Optimum, d.h.:

Motivation

- Bisher zwei gute einfache Heuristiken
- Frage: wie gut können wir das Scheduling Problem auf identischen Maschinen approximieren?
- Frage: gibt es da ggf. eine untere Schranke, oder kommen wir beliebig nah an das Optimum heran?
- Antwort: wir kommen beliebig nah an das Optimum, d.h.:
  - Gegeben sei ein beliebiges kleines konstantes Epsilon ε.

Motivation

- Bisher zwei gute einfache Heuristiken
- Frage: wie gut können wir das Scheduling Problem auf identischen Maschinen approximieren?
- Frage: gibt es da ggf. eine untere Schranke, oder kommen wir beliebig nah an das Optimum heran?
- Antwort: wir kommen beliebig nah an das Optimum, d.h.:
  - Gegeben sei ein beliebiges kleines konstantes Epsilon ε.
  - Dann gibt es einen Polynomzeit-Algorithmus, der bis auf einen Faktor von $1+\varepsilon$ approximiert.

Motivation

- Bisher zwei gute einfache Heuristiken
- Frage: wie gut können wir das Scheduling Problem auf identischen Maschinen approximieren?
- Frage: gibt es da ggf. eine untere Schranke, oder kommen wir beliebig nah an das Optimum heran?
- Antwort: wir kommen beliebig nah an das Optimum, d.h.:
  - Gegeben sei ein beliebiges kleines konstantes Epsilon ε.
  - Dann gibt es einen Polynomzeit-Algorithmus, der bis auf einen Faktor von $1 + \varepsilon$ approximiert.
  - Wichtig hier: die Laufzeit hängt von ε ab.

Definition

Definition (PTAS)

Ein Optimierungsproblem Π hat ein polynomielles Approximationsschema, falls es für jedes konstante $\varepsilon > 0$ eine $(1+\varepsilon)$-Approximation in polynomieller Zeit berechnet werden kann (Bei einem Maximierungsproblem: $(1-\varepsilon)$).

Definition

Definition (PTAS)

Ein Optimierungsproblem Π hat ein polynomielles Approximationsschema, falls es für jedes konstante $\varepsilon > 0$ eine $(1+\varepsilon)$-Approximation in polynomieller Zeit berechnet werden kann (Bei einem Maximierungsproblem: $(1-\varepsilon)$).

Definition (FPTAS)

Ein Optimierungsproblem Π hat ein voll polynomielles Approximationsschema, falls es für jedes konstante $\varepsilon > 0$ eine $(1+\varepsilon)$-Approximation in polynomieller Zeit in der Eingabegröße und $1/\varepsilon$ berechnet werden kann (Bei einem Maximierungsproblem: $(1-\varepsilon)$).

Definition

Definition (PTAS)

Ein Optimierungsproblem Π hat ein polynomielles Approximationsschema, falls es für jedes konstante $\varepsilon > 0$ eine $(1+\varepsilon)$-Approximation in polynomieller Zeit berechnet werden kann (Bei einem Maximierungsproblem: $(1-\varepsilon)$).

Definition (FPTAS)

Ein Optimierungsproblem Π hat ein voll polynomielles Approximationsschema, falls es für jedes konstante $\varepsilon > 0$ eine $(1+\varepsilon)$-Approximation in polynomieller Zeit in der Eingabegröße und $1/\varepsilon$ berechnet werden kann (Bei einem Maximierungsproblem: $(1-\varepsilon)$).

- Das Makespan Scheduling Problem ist stark $\mathcal{NP}$-hart.

Definition

Definition (PTAS)

Ein Optimierungsproblem Π hat ein polynomielles Approximationsschema, falls es für jedes konstante $\varepsilon > 0$ eine $(1+\varepsilon)$-Approximation in polynomieller Zeit berechnet werden kann (Bei einem Maximierungsproblem: $(1-\varepsilon)$).

Definition (FPTAS)

Ein Optimierungsproblem Π hat ein voll polynomielles Approximationsschema, falls es für jedes konstante $\varepsilon > 0$ eine $(1+\varepsilon)$-Approximation in polynomieller Zeit in der Eingabegröße und $1/\varepsilon$ berechnet werden kann (Bei einem Maximierungsproblem: $(1-\varepsilon)$).

- Das Makespan Scheduling Problem ist stark $\mathcal{NP}$-hart.
- Daher kann das Makespan Scheduling Problem kein FPTAS haben.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.
  - Herunterskaliert und geringfügig vergrößert.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.
  - Herunterskaliert und geringfügig vergrößert.
  - Dabei wird ein Fehler von ε in Kauf genommen.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.
  - Herunterskaliert und geringfügig vergrößert.
  - Dabei wird ein Fehler von ε in Kauf genommen.
  - Dadurch wird der obige Algorithmus für Bin Packing anwendbar.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.
  - Herunterskaliert und geringfügig vergrößert.
  - Dabei wird ein Fehler von ε in Kauf genommen.
  - Dadurch wird der obige Algorithmus für Bin Packing anwendbar.
- Die kleinen Jobs aus K werden in der zweiten Phase mittels Heuristik verteilt.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.
  - Herunterskaliert und geringfügig vergrößert.
  - Dabei wird ein Fehler von ε in Kauf genommen.
  - Dadurch wird der obige Algorithmus für Bin Packing anwendbar.
- Die kleinen Jobs aus K werden in der zweiten Phase mittels Heuristik verteilt.
  - Da diese klein sind, werden sie nur einen kleinen Fehler erzeugen.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.
  - Herunterskaliert und geringfügig vergrößert.
  - Dabei wird ein Fehler von ε in Kauf genommen.
  - Dadurch wird der obige Algorithmus für Bin Packing anwendbar.
- Die kleinen Jobs aus K werden in der zweiten Phase mittels Heuristik verteilt.
  - Da diese klein sind, werden sie nur einen kleinen Fehler erzeugen.
- Problem dabei:

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.
  - Herunterskaliert und geringfügig vergrößert.
  - Dabei wird ein Fehler von ε in Kauf genommen.
  - Dadurch wird der obige Algorithmus für Bin Packing anwendbar.
- Die kleinen Jobs aus K werden in der zweiten Phase mittels Heuristik verteilt.
  - Da diese klein sind, werden sie nur einen kleinen Fehler erzeugen.
- Problem dabei:
  - Aufteilung nutzt das optimale Makespan.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.
  - Herunterskaliert und geringfügig vergrößert.
  - Dabei wird ein Fehler von ε in Kauf genommen.
  - Dadurch wird der obige Algorithmus für Bin Packing anwendbar.
- Die kleinen Jobs aus K werden in der zweiten Phase mittels Heuristik verteilt.
  - Da diese klein sind, werden sie nur einen kleinen Fehler erzeugen.
- Problem dabei:
  - Aufteilung nutzt das optimale Makespan.
  - Skalierung nutzt das optimale Makespan.

Aufbau der Idee

- Teile die Jobs in große und kleine Jobs auf: G und K.
- Die Gewichte der großen Jobs werden verändert.
  - Herunterskaliert und geringfügig vergrößert.
  - Dabei wird ein Fehler von ε in Kauf genommen.
  - Dadurch wird der obige Algorithmus für Bin Packing anwendbar.
- Die kleinen Jobs aus K werden in der zweiten Phase mittels Heuristik verteilt.
  - Da diese klein sind, werden sie nur einen kleinen Fehler erzeugen.
- Problem dabei:
  - Aufteilung nutzt das optimale Makespan.
  - Skalierung nutzt das optimale Makespan.
- Lösung: Approximiere das optimale Makespan mittels Binärsuche.

Algorithmus

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.

Algorithmus

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1:

Algorithmus

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1:
   1. Betrachte die großen Jobs: $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$.

Algorithmus

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1:
   1. Betrachte die großen Jobs: $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$.
   2. Skaliere die Größe der Jobs aus G:

$$p'_i = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil$$

Algorithmus

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1:
   1. Betrachte die großen Jobs: $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$.
   2. Skaliere die Größe der Jobs aus G:
   $$p_i' = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil$$
   3. Bestimme Schedule mit Jobgrößen p_i' mit Makespan
   $$Z' = \left\lfloor (1+\varepsilon)\frac{1}{\varepsilon^2} \right\rfloor .$$

Algorithmus

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1:
   1. Betrachte die großen Jobs: $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$.
   2. Skaliere die Größe der Jobs aus G:
   $$p_i' = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil$$
   3. Bestimme Schedule mit Jobgrößen p_i' mit Makespan
   $$Z' = \left\lfloor (1+\varepsilon) \frac{1}{\varepsilon^2} \right\rfloor .$$
3. Phase 2:

Algorithmus

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1:
   1. Betrachte die großen Jobs: $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$.
   2. Skaliere die Größe der Jobs aus G:
   $$p_i' = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil$$
   3. Bestimme Schedule mit Jobgrößen p_i' mit Makespan
   $$Z' = \left\lfloor (1+\varepsilon)\frac{1}{\varepsilon^2} \right\rfloor .$$
3. Phase 2:
   1. Betrachte die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$.

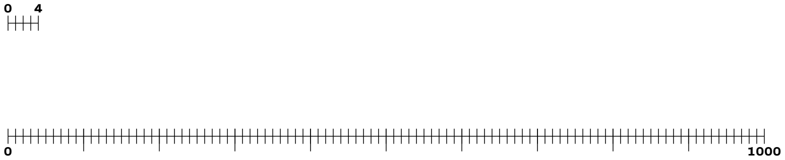
Algorithmus

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1:
   1. Betrachte die großen Jobs: $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$.
   2. Skaliere die Größe der Jobs aus G:
   $$p_i' = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil$$
   3. Bestimme Schedule mit Jobgrößen p_i' mit Makespan
   $$Z' = \left\lfloor (1+\varepsilon)\frac{1}{\varepsilon^2} \right\rfloor.$$
3. Phase 2:
   1. Betrachte die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$.
   2. Verteile die Jobs aus K nach der LL Heuristik.

Beispiel zur Skalierung (1/2)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

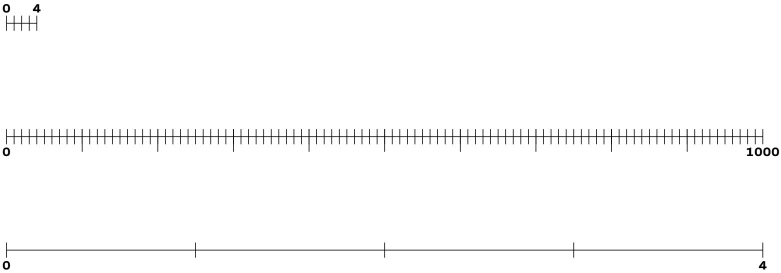
- $Z = 1000$ und $\varepsilon = 1/2$.



Beispiel zur Skalierung (1/2)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

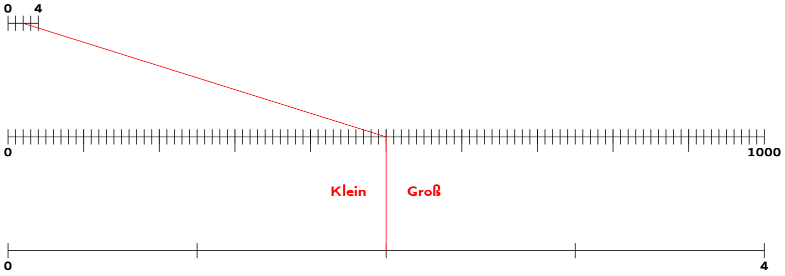
- $Z = 1000$ und $\varepsilon = 1/2$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 4\}$



Beispiel zur Skalierung (1/2)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

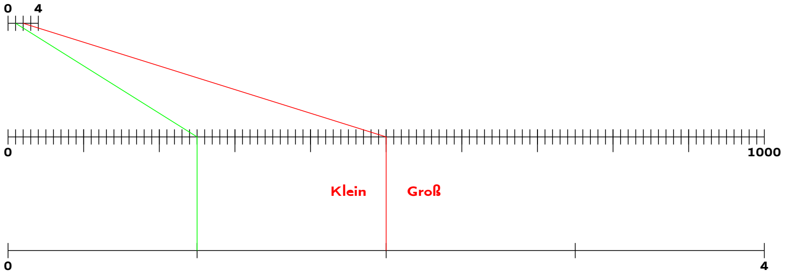
- $Z = 1000$ und $\varepsilon = 1/2$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 4\}$



Beispiel zur Skalierung (1/2)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

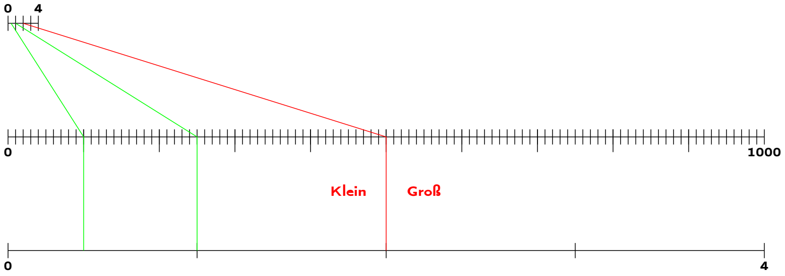
- $Z = 1000$ und $\varepsilon = 1/2$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 4\}$



Beispiel zur Skalierung (1/2)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z)\rceil$

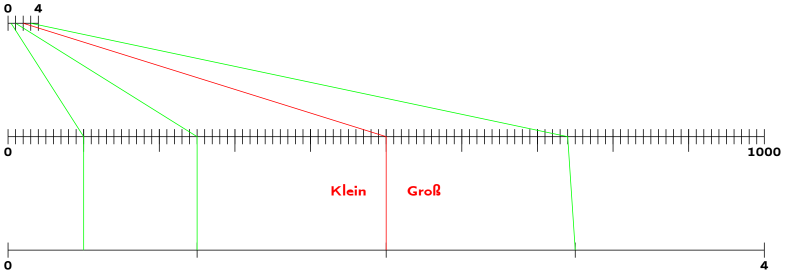
- $Z = 1000$ und $\varepsilon = 1/2$.
- Skalierung: $\{0,\ldots,1000\} \mapsto \{0,\ldots,4\}$



Beispiel zur Skalierung (1/2)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z)\rceil$

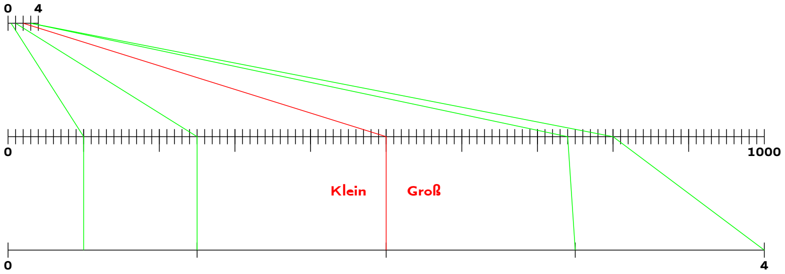
- $Z = 1000$ und $\varepsilon = 1/2$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 4\}$



Beispiel zur Skalierung (1/2)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z)\rceil$

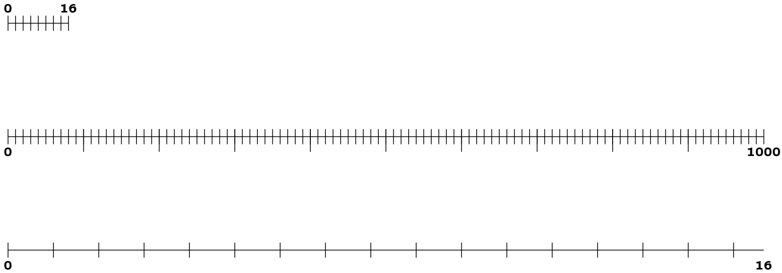
- $Z = 1000$ und $\varepsilon = 1/2$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 4\}$



Beispiel zur Skalierung (1/4)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

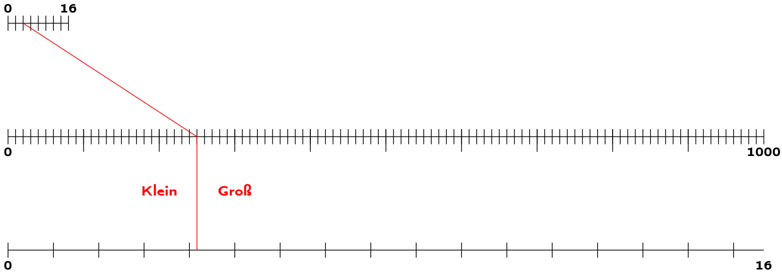
- $Z = 1000$ und $\varepsilon = 1/4$.



Beispiel zur Skalierung (1/4)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z)\rceil$

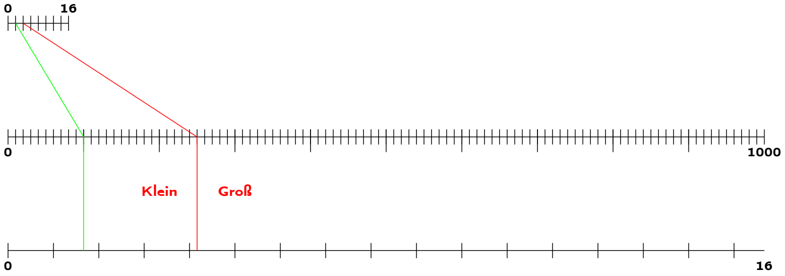
- $Z = 1000$ und $\varepsilon = 1/4$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 16\}$



Beispiel zur Skalierung (1/4)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

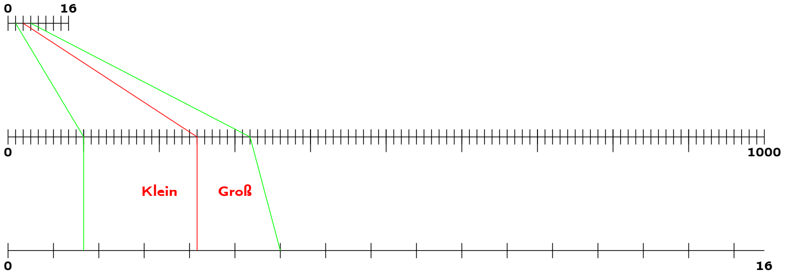
- $Z = 1000$ und $\varepsilon = 1/4$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 16\}$



Beispiel zur Skalierung (1/4)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

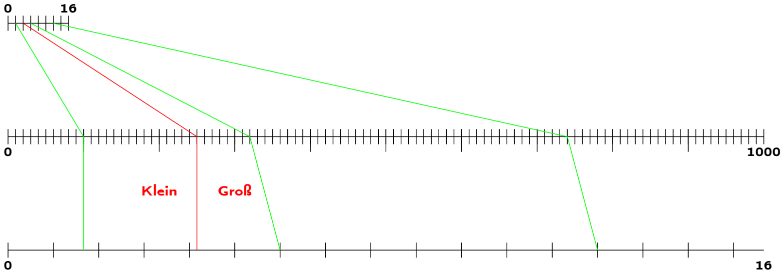
- $Z = 1000$ und $\varepsilon = 1/4$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 16\}$



Beispiel zur Skalierung (1/4)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

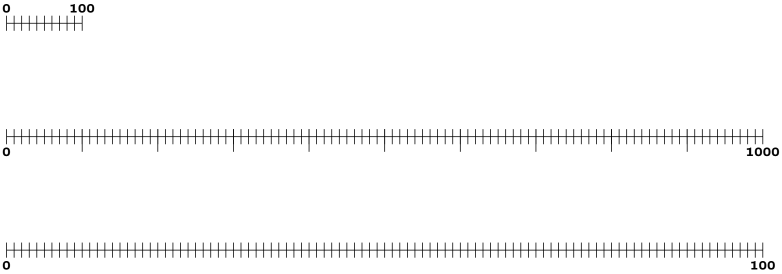
- $Z = 1000$ und $\varepsilon = 1/4$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 16\}$



Beispiel zur Skalierung (1/10)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

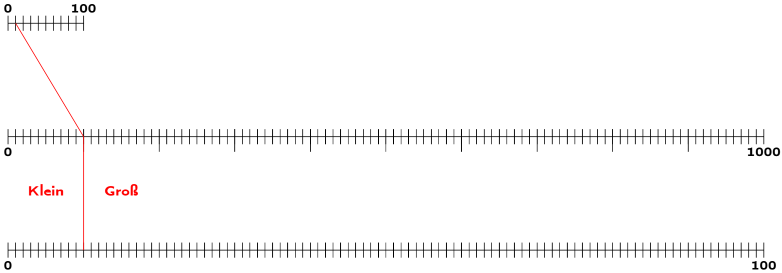
- $Z = 1000$ und $\varepsilon = 1/10$.



Beispiel zur Skalierung (1/10)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

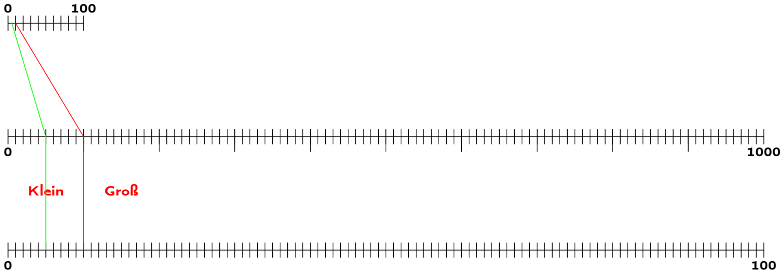
- $Z = 1000$ und $\varepsilon = 1/10$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 100\}$



Beispiel zur Skalierung (1/10)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z) \rceil$

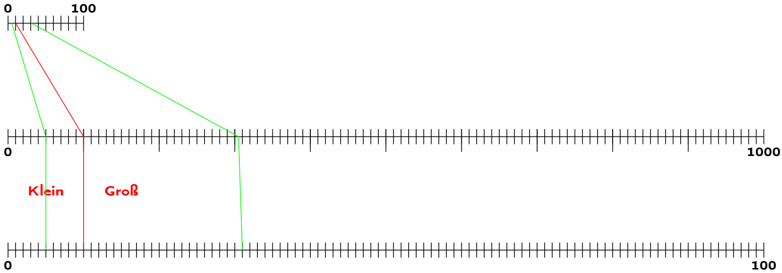
- $Z = 1000$ und $\varepsilon = 1/10$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 100\}$



Beispiel zur Skalierung (1/10)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z)\rceil$

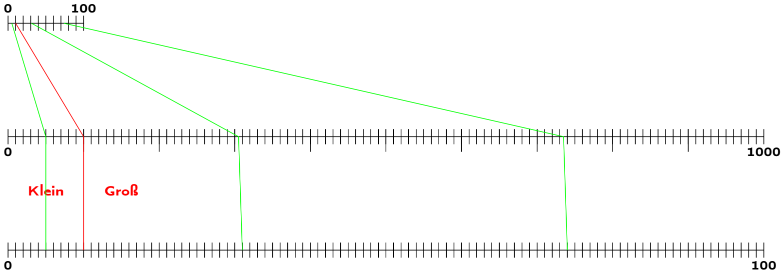
- $Z = 1000$ und $\varepsilon = 1/10$.
- Skalierung: $\{0, \ldots, 1000\} \mapsto \{0, \ldots, 100\}$



Beispiel zur Skalierung (1/10)

Klein: $p_i \leqslant \varepsilon Z$
Groß: $p_i > \varepsilon Z$
Skalierung: $\lceil p_i/(\varepsilon^2 Z)\rceil$

- $Z = 1000$ und $\varepsilon = 1/10$.
- Skalierung: $\{0,\ldots,1000\} \mapsto \{0,\ldots,100\}$



Phase 1 (Beispiel zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Sei $Z = 1000$ und $\varepsilon = 0.1$.

Phase 1 (Beispiel zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Sei $Z = 1000$ und $\varepsilon = 0.1$.
- Dann ist $p_i'' = \frac{p_i}{\varepsilon^2 Z} = \frac{p_i}{\varepsilon^2 1000} = \frac{p_i}{10}$.

Phase 1 (Beispiel zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z)\rceil$

- Sei $Z = 1000$ und $\varepsilon = 0.1$.
- Dann ist $p_i'' = \frac{p_i}{\varepsilon^2 Z} = \frac{p_i}{\varepsilon^2 1000} = \frac{p_i}{10}$.
- Dann ist $p_i' = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil = \left\lceil \frac{p_i}{\varepsilon^2 1000} \right\rceil = \left\lceil \frac{p_i}{10} \right\rceil$.

Phase 1 (Beispiel zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Sei $Z = 1000$ und $\varepsilon = 0.1$.
- Dann ist $p_i'' = \frac{p_i}{\varepsilon^2 Z} = \frac{p_i}{\varepsilon^2 1000} = \frac{p_i}{10}$.
- Dann ist $p_i' = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil = \left\lceil \frac{p_i}{\varepsilon^2 1000} \right\rceil = \left\lceil \frac{p_i}{10} \right\rceil$.
- Sei beispielsweise $p_i = 101$.

Phase 1 (Beispiel zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Sei $Z = 1000$ und $\varepsilon = 0.1$.
- Dann ist $p_i'' = \frac{p_i}{\varepsilon^2 Z} = \frac{p_i}{\varepsilon^2 1000} = \frac{p_i}{10}$.
- Dann ist $p_i' = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil = \left\lceil \frac{p_i}{\varepsilon^2 1000} \right\rceil = \left\lceil \frac{p_i}{10} \right\rceil$.
- Sei beispielsweise $p_i = 101$.
  - Beispiel: $p_i'' = \frac{101}{\varepsilon^2 Z} = \frac{101}{10} = 10.1$.

Phase 1 (Beispiel zur Skalierung)

$p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Sei $Z = 1000$ und $\varepsilon = 0.1$.
- Dann ist $p''_i = \frac{p_i}{\varepsilon^2 Z} = \frac{p_i}{\varepsilon^2 1000} = \frac{p_i}{10}$.
- Dann ist $p'_i = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil = \left\lceil \frac{p_i}{\varepsilon^2 1000} \right\rceil = \left\lceil \frac{p_i}{10} \right\rceil$.
- Sei beispielsweise $p_i = 101$.
  - Beispiel: $p''_i = \frac{101}{\varepsilon^2 Z} = \frac{101}{10} = 10.1$.
  - Beispiel: $p'_i = \left\lceil \frac{101}{\varepsilon^2 Z} \right\rceil = \left\lceil \frac{101}{10} \right\rceil = \lceil 10.1 \rceil = 11$.

Phase 1 (Beispiel zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Sei $Z = 1000$ und $\varepsilon = 0.1$.
- Dann ist $p_i'' = \frac{p_i}{\varepsilon^2 Z} = \frac{p_i}{\varepsilon^2 1000} = \frac{p_i}{10}$.
- Dann ist $p_i' = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil = \left\lceil \frac{p_i}{\varepsilon^2 1000} \right\rceil = \left\lceil \frac{p_i}{10} \right\rceil$.
- Sei beispielsweise $p_i = 101$.
  - Beispiel: $p_i'' = \frac{101}{\varepsilon^2 Z} = \frac{101}{10} = 10.1$.
  - Beispiel: $p_i' = \left\lceil \frac{101}{\varepsilon^2 Z} \right\rceil = \left\lceil \frac{101}{10} \right\rceil = \lceil 10.1 \rceil = 11$.
- Damit ist für dieses Beispiel der relative Rundungsfehler:

$$\frac{p_i' - p_i''}{p_i''} = \frac{11 - 10.1}{10.1} = \frac{0.9}{10.1} \leqslant 0.08911 \leqslant \varepsilon$$

Phase 1 (Beispiel zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Sei $Z = 1000$ und $\varepsilon = 0.1$.
- Dann ist $p_i'' = \frac{p_i}{\varepsilon^2 Z} = \frac{p_i}{\varepsilon^2 1000} = \frac{p_i}{10}$.
- Dann ist $p_i' = \left\lceil \frac{p_i}{\varepsilon^2 Z} \right\rceil = \left\lceil \frac{p_i}{\varepsilon^2 1000} \right\rceil = \left\lceil \frac{p_i}{10} \right\rceil$.
- Sei beispielsweise $p_i = 101$.
  - Beispiel: $p_i'' = \frac{101}{\varepsilon^2 Z} = \frac{101}{10} = 10.1$.
  - Beispiel: $p_i' = \left\lceil \frac{101}{\varepsilon^2 Z} \right\rceil = \left\lceil \frac{101}{10} \right\rceil = \lceil 10.1 \rceil = 11$.
- Damit ist für dieses Beispiel der relative Rundungsfehler:

$$\frac{p_i' - p_i''}{p_i''} = \frac{11 - 10.1}{10.1} = \frac{0.9}{10.1} \leqslant 0.08911 \leqslant \varepsilon$$

- Das folgende Lemma zeigt, dass das im Allgemeinen gilt.

Phase 1 (Beweis zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z)\rceil$

Lemma

Der relative Rundungsfehler $\frac{p_i' - p_i''}{p_i''}$ *ist höchstens* ε.

Beweis

Phase 1 (Beweis zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Der relative Rundungsfehler $\frac{p_i' - p_i''}{p_i''}$ *ist höchstens* ε.

Beweis

Phase 1 (Beweis zur Skalierung)

$p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Der relative Rundungsfehler $\frac{p'_i - p''_i}{p''_i}$ *ist höchstens* ε.

Beweis

- Sei $i \in G$ ein großer Job, d.h. $p_i \geqslant \varepsilon Z$.

Phase 1 (Beweis zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Der relative Rundungsfehler $\frac{p_i' - p_i''}{p_i''}$ *ist höchstens* ε.

Beweis

- Sei $i \in G$ ein großer Job, d.h. $p_i \geqslant \varepsilon Z$.
- Damit gilt:

$$p_i'' \geqslant \frac{\varepsilon Z}{\varepsilon^2 Z} = \frac{1}{\varepsilon}.$$

Phase 1 (Beweis zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Der relative Rundungsfehler $\frac{p_i' - p_i''}{p_i''}$ *ist höchstens* ε.

Beweis

- Sei $i \in G$ ein großer Job, d.h. $p_i \geqslant \varepsilon Z$.
- Damit gilt:

$$p_i'' \geqslant \frac{\varepsilon Z}{\varepsilon^2 Z} = \frac{1}{\varepsilon}.$$

- Daraus und wegen $p_i' - p_i'' \leqslant 1$ folgt nun:

$$\frac{p_i' - p_i''}{p_i''} \leqslant \frac{1}{1/\varepsilon} = \varepsilon.$$

Phase 1 (Beweis zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Der relative Rundungsfehler $\frac{p_i' - p_i''}{p_i''}$ *ist höchstens* ε*.*

Beweis

- Sei $i \in G$ ein großer Job, d.h. $p_i \geqslant \varepsilon Z$.
- Damit gilt:

$$p_i'' \geqslant \frac{\varepsilon Z}{\varepsilon^2 Z} = \frac{1}{\varepsilon}.$$

- Daraus und wegen $p_i' - p_i'' \leqslant 1$ folgt nun:

$$\frac{p_i' - p_i''}{p_i''} \leqslant \frac{1}{1/\varepsilon} = \varepsilon.$$

Phase 1 (Beweis zur Skalierung)

$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Der relative Rundungsfehler $\frac{p_i' - p_i''}{p_i''}$ ist höchstens ε.

Beweis

- Sei $i \in G$ ein großer Job, d.h. $p_i \geqslant \varepsilon Z$.
- Damit gilt:

$$p_i'' \geqslant \frac{\varepsilon Z}{\varepsilon^2 Z} = \frac{1}{\varepsilon}.$$

- Daraus und wegen $p_i' - p_i'' \leqslant 1$ folgt nun:

$$\frac{p_i' - p_i''}{p_i''} \leqslant \frac{1}{1/\varepsilon} = \varepsilon.$$

Folgerung:

Das Aufrunden der skalierten Größen verändert diese um maximal einen Faktor von $1 + \varepsilon$.

Phase 1 (Beweis der Existenz eines Schedule)

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

Lemma

Auch für die veränderten Werte p_i' und Z' gibt es ein korrektes Schedule.

Beweis:

Phase 1 (Beweis der Existenz eines Schedule)

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

Lemma

Auch für die veränderten Werte p_i' und Z' gibt es ein korrektes Schedule.

Beweis:

Phase 1 (Beweis der Existenz eines Schedule)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Auch für die veränderten Werte p_i' und Z' gibt es ein korrektes Schedule.

Beweis:

- Für p_i und Z gibt es ein Schedule.

Phase 1 (Beweis der Existenz eines Schedule)

$Z' = \lfloor (\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

Lemma

Auch für die veränderten Werte p_i' und Z' gibt es ein korrektes Schedule.

Beweis:

- Für p_i und Z gibt es ein Schedule.
- Damit gibt es auch ein Schedule für:

$$p_i'' = p_i/(\varepsilon^2 Z) \text{ und } Z'' = Z/(\varepsilon^2 Z) = 1/\varepsilon^2.$$

Phase 1 (Beweis der Existenz eines Schedule)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Auch für die veränderten Werte p_i' und Z' gibt es ein korrektes Schedule.

Beweis:

- Für p_i und Z gibt es ein Schedule.
- Damit gibt es auch ein Schedule für:

$$p_i'' = p_i/(\varepsilon^2 Z) \text{ und } Z'' = Z/(\varepsilon^2 Z) = 1/\varepsilon^2.$$

- Durch das Aufrunden erhöht sich der Makespan höchstens um einen Faktor von $(1+\varepsilon)$. Also gibt es ein Schedule für:

$$p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil \text{ und } Z'' = (1+\varepsilon)/\varepsilon^2.$$

Phase 1 (Beweis der Existenz eines Schedule)

$Z' = \lfloor(\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}}Z)\rceil$

Lemma

Auch für die veränderten Werte p_i' und Z' gibt es ein korrektes Schedule.

Beweis:

- Für p_i und Z gibt es ein Schedule.
- Damit gibt es auch ein Schedule für:

$$p_i'' = p_i/(\varepsilon^2 Z) \text{ und } Z'' = Z/(\varepsilon^2 Z) = 1/\varepsilon^2.$$

- Durch das Aufrunden erhöht sich der Makespan höchstens um einen Faktor von $(1+\varepsilon)$. Also gibt es ein Schedule für:

$$p_i' = \lceil p_i/(\varepsilon^2 Z)\rceil \text{ und } Z'' = (1+\varepsilon)/\varepsilon^2.$$

- Durch das Aufrunden $p' = \lceil p'' \rceil$ ist der Makespan ganzzahlig.

Phase 1 (Beweis der Existenz eines Schedule)

$Z' = \lfloor(1+\varepsilon)/(\varepsilon^2)\rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z)\rceil$

Lemma

Auch für die veränderten Werte p'_i und Z' gibt es ein korrektes Schedule.

Beweis:

- Für p_i und Z gibt es ein Schedule.
- Damit gibt es auch ein Schedule für:

$$p''_i = p_i/(\varepsilon^2 Z) \text{ und } Z'' = Z/(\varepsilon^2 Z) = 1/\varepsilon^2.$$

- Durch das Aufrunden erhöht sich der Makespan höchstens um einen Faktor von $(1+\varepsilon)$. Also gibt es ein Schedule für:

$$p'_i = \lceil p_i/(\varepsilon^2 Z)\rceil \text{ und } Z'' = (1+\varepsilon)/\varepsilon^2.$$

- Durch das Aufrunden $p' = \lceil p'' \rceil$ ist der Makespan ganzzahlig.
- Damit gibt es ein Schedule für:

$$p'_i = \lceil p_i/(\varepsilon^2 Z)\rceil \text{ und } Z' = \lfloor(1+\varepsilon)/\varepsilon^2\rfloor.$$

Phase 1 (Anwendung von Bin Packing)

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

Lemma

Die Werte k und b für die Anwendung des Bin Packing Algorithmus sind:

$$k = \left\lceil \frac{1}{\varepsilon^2} \right\rceil \text{ und } b = Z' = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor .$$

Beweis:

Phase 1 (Anwendung von Bin Packing)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Die Werte k und b für die Anwendung des Bin Packing Algorithmus sind:

$$k = \left\lceil \frac{1}{\varepsilon^2} \right\rceil \text{ und } b = Z' = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor.$$

Beweis:

Phase 1 (Anwendung von Bin Packing)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Die Werte k und b für die Anwendung des Bin Packing Algorithmus sind:

$$k = \left\lceil \frac{1}{\varepsilon^2} \right\rceil \text{ und } b = Z' = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor .$$

Beweis:

- Die maximale Jobgröße ist Z.

Phase 1 (Anwendung von Bin Packing)

$Z' = \lfloor(\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

Lemma

Die Werte k und b für die Anwendung des Bin Packing Algorithmus sind:

$$k = \left\lceil \frac{1}{\varepsilon^2} \right\rceil \text{ und } b = Z' = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor.$$

Beweis:

- Die maximale Jobgröße ist Z.
- Damit ist die maximale Größe nach der Skalierung: $\lceil Z/(\varepsilon^2 Z)\rceil = \lceil 1/\varepsilon^2\rceil$.

Phase 1 (Anwendung von Bin Packing)

$Z' = \lfloor(1+\varepsilon)/(\varepsilon^2)\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z)\rceil$

Lemma

Die Werte k und b für die Anwendung des Bin Packing Algorithmus sind:

$$k = \left\lceil \frac{1}{\varepsilon^2} \right\rceil \text{ und } b = Z' = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor.$$

Beweis:

- Die maximale Jobgröße ist Z.
- Damit ist die maximale Größe nach der Skalierung: $\lceil Z/(\varepsilon^2 Z)\rceil = \lceil 1/\varepsilon^2\rceil$.
- Dass $b = Z'$ gilt, ist klar.

Phase 1 (Anwendung von Bin Packing)

$Z' = \lfloor(\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}}Z)\rceil$

Lemma

Die Werte k und b für die Anwendung des Bin Packing Algorithmus sind:

$$k = \left\lceil \frac{1}{\varepsilon^2} \right\rceil \textit{ und } b = Z' = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor.$$

Beweis:

- Die maximale Jobgröße ist Z.
- Damit ist die maximale Größe nach der Skalierung: $\lceil Z/(\varepsilon^2 Z)\rceil = \lceil 1/\varepsilon^2\rceil$.
- Dass $b = Z'$ gilt, ist klar.

Phase 1 (Anwendung von Bin Packing)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Die Werte k und b für die Anwendung des Bin Packing Algorithmus sind:

$$k = \left\lceil \frac{1}{\varepsilon^2} \right\rceil \text{ und } b = Z' = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor .$$

Beweis:

- Die maximale Jobgröße ist Z.
- Damit ist die maximale Größe nach der Skalierung: $\lceil Z/(\varepsilon^2 Z) \rceil = \lceil 1/\varepsilon^2 \rceil$.
- Dass $b = Z'$ gilt, ist klar.

Folgerung

Die Laufzeit ist: $O((n+1)^k \cdot (b+1)^k / k!) = O(n^{\lceil 1/\varepsilon^2 \rceil})$.

Approximationsfaktor

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

Approximationsfaktor

$Z' = \lfloor(\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Betrachte Jobs aus G (Phase 1):

Approximationsfaktor

$Z' = \lfloor(\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1 + \varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Betrachte Jobs aus G (Phase 1):
  - Der Makespan Z' kann höchstens um einen Faktor von $\varepsilon^2 \cdot Z$ größer sein.

Approximationsfaktor

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Betrachte Jobs aus G (Phase 1):
  - Der Makespan Z' kann höchstens um einen Faktor von $\varepsilon^2 \cdot Z$ größer sein.
  - Damit ist der Makespan für die Jobs aus G:

$$Z' \cdot \varepsilon^2 \cdot Z = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor \cdot \varepsilon^2 \cdot Z \leqslant (1+\varepsilon)Z.$$

Approximationsfaktor

$Z' = \lfloor(\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1 + \varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Betrachte Jobs aus G (Phase 1):
  - Der Makespan Z' kann höchstens um einen Faktor von $\varepsilon^2 \cdot Z$ größer sein.
  - Damit ist der Makespan für die Jobs aus G:

$$Z' \cdot \varepsilon^2 \cdot Z = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor \cdot \varepsilon^2 \cdot Z \leqslant (1+\varepsilon)Z.$$

  - Das ist eine $(1 + \varepsilon)$-Approximation.

Approximationsfaktor

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1 + \varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Betrachte Jobs aus G (Phase 1):
  - Der Makespan Z' kann höchstens um einen Faktor von $\varepsilon^2 \cdot Z$ größer sein.
  - Damit ist der Makespan für die Jobs aus G:

$$Z' \cdot \varepsilon^2 \cdot Z = \left\lfloor \frac{1+\varepsilon}{\varepsilon^2} \right\rfloor \cdot \varepsilon^2 \cdot Z \leqslant (1+\varepsilon)Z.$$

  - Das ist eine $(1 + \varepsilon)$-Approximation.
- Betrachte Jobs aus K (Phase 2) auf der nächsten Folie.

Approximationsfaktor

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

Approximationsfaktor

$Z' = \lfloor(\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Jobs aus G (Phase 1) sind abgeschätzt.

Approximationsfaktor

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Jobs aus G (Phase 1) sind abgeschätzt.
- Betrachte Jobs aus K (Phase 2):

Approximationsfaktor

$Z' = \lfloor(\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}}Z)\rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Jobs aus G (Phase 1) sind abgeschätzt.
- Betrachte Jobs aus K (Phase 2):
  - Falls die Jobs aus K den Makespan nicht erhöhen, gilt die Behauptung.

Approximationsfaktor

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Jobs aus G (Phase 1) sind abgeschätzt.
- Betrachte Jobs aus K (Phase 2):
  - Falls die Jobs aus K den Makespan nicht erhöhen, gilt die Behauptung.
  - Sei i der Job, der einen erhöhten Makespan von L erzeugt.

Approximationsfaktor

$Z' = \lfloor(\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}}Z)\rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Jobs aus G (Phase 1) sind abgeschätzt.
- Betrachte Jobs aus K (Phase 2):
  - Falls die Jobs aus K den Makespan nicht erhöhen, gilt die Behauptung.
  - Sei i der Job, der einen erhöhten Makespan von L erzeugt.
  - Wegen der LL Heuristik haben nach der Platzierung von K alle Maschinen eine Last von mindestens $L - p_i$.

Approximationsfaktor

$Z' = \lfloor(\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Jobs aus G (Phase 1) sind abgeschätzt.
- Betrachte Jobs aus K (Phase 2):
  - Falls die Jobs aus K den Makespan nicht erhöhen, gilt die Behauptung.
  - Sei i der Job, der einen erhöhten Makespan von L erzeugt.
  - Wegen der LL Heuristik haben nach der Platzierung von K alle Maschinen eine Last von mindestens $L - p_i$.
  - Damit gilt für den optimalen Makespan: $Z \geqslant L - p_i$.

Approximationsfaktor

$Z' = \lfloor(\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

Lemma

Der obige Algorithmus bestimmt eine $(1+\varepsilon)$-Approximation für einen minimalen Makespan.

Beweis:

- Jobs aus G (Phase 1) sind abgeschätzt.
- Betrachte Jobs aus K (Phase 2):
  - Falls die Jobs aus K den Makespan nicht erhöhen, gilt die Behauptung.
  - Sei i der Job, der einen erhöhten Makespan von L erzeugt.
  - Wegen der LL Heuristik haben nach der Platzierung von K alle Maschinen eine Last von mindestens $L - p_i$.
  - Damit gilt für den optimalen Makespan: $Z \geqslant L - p_i$.
  - Folglich: $L \leqslant Z + p_i \leqslant (1+\varepsilon) \cdot Z$.

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor(\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z)\rceil$.

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p_i' mit Makespan

$$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \dots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p'_i mit Makespan

$$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$

3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \dots, n\} \mid p_i \leqslant \varepsilon Z\}$:

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p_i' mit Makespan
   $$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$
3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p_i' mit Makespan

$$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$

3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

- Kann es das obige Orakel geben?

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p'_i mit Makespan

$$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$

3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

- Kann es das obige Orakel geben?
- Nein, denn dann könnten wir das Scheduling Problem effizient lösen.

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor(1+\varepsilon)/(\varepsilon^2)\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z)\rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p_i' = \lceil p_i/(\varepsilon^2 Z)\rceil$.
   2. Bestimme Schedule mit Jobgrößen p_i' mit Makespan
   $$Z' = \lfloor(1+\varepsilon)/\varepsilon^2\rfloor.$$
3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

- Kann es das obige Orakel geben?
- Nein, denn dann könnten wir das Scheduling Problem effizient lösen.
- Da wir ja schon approximieren, brauchen wir auch Z auch nicht genau zu bestimmen.

Das Orakel (Erinnerung und Vorüberlegungen)

$Z' = \lfloor(1+\varepsilon)/(\varepsilon^2)\rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z)\rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z)\rceil$.
   2. Bestimme Schedule mit Jobgrößen p'_i mit Makespan
      $$Z' = \lfloor(1+\varepsilon)/\varepsilon^2\rfloor.$$
3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

- Kann es das obige Orakel geben?
- Nein, denn dann könnten wir das Scheduling Problem effizient lösen.
- Da wir ja schon approximieren, brauchen wir auch Z auch nicht genau zu bestimmen.
- Idee: versuche Halbierungssuche mit Hilfe des obigen Algorithmus.

Das Orakel (Aufbau der Idee)

$Z' = \lfloor (\mathbf{1} + \varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.

Das Orakel (Aufbau der Idee)

$Z' = \lfloor (\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:

Das Orakel (Aufbau der Idee)

$Z' = \lfloor(\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1,2,\ldots,n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z)\rceil$.

Das Orakel (Aufbau der Idee)

$Z' = \lfloor(1+\varepsilon)/(\varepsilon^2)\rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z)\rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z)\rceil$.
   2. Bestimme Schedule mit Jobgrößen p'_i mit Makespan

$$Z' = \lfloor(1+\varepsilon)/\varepsilon^2\rfloor.$$

Das Orakel (Aufbau der Idee)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1,2,\ldots,n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p'_i mit Makespan

$$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$

3. Phase 2, die kleinen Jobs: $K = \{i \in \{1,2,\ldots,n\} \mid p_i \leqslant \varepsilon Z\}$:

Das Orakel (Aufbau der Idee)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

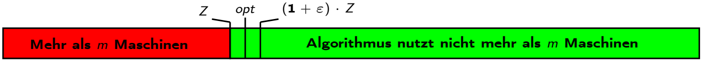
1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p'_i mit Makespan
   $$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$
3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

Das Orakel (Aufbau der Idee)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1,2,\ldots,n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p'_i mit Makespan
   $$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$
3. Phase 2, die kleinen Jobs: $K = \{i \in \{1,2,\ldots,n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

- Es gibt $Z^* \leqslant opt$:
  das Bin Packing ist lösbar, falls $Z \geqslant Z^*$ gewählt wurde.



Das Orakel (Aufbau der Idee)

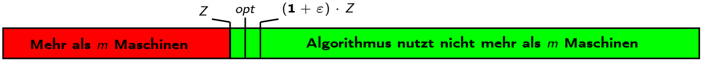
$Z' = \lfloor(1+\varepsilon)/(\varepsilon^2)\rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z)\rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p_i' = \lceil p_i/(\varepsilon^2 Z)\rceil$.
   2. Bestimme Schedule mit Jobgrößen p_i' mit Makespan

$$Z' = \lfloor(1+\varepsilon)/\varepsilon^2\rfloor.$$

3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

- Es gibt $Z^* \leqslant opt$:
  das Bin Packing ist lösbar, falls $Z \geqslant Z^*$ gewählt wurde.



- Modifiziere Algorithmus so, dass er zu kleine Werte für Z erkennt.

Das Orakel (Aufbau der Idee)

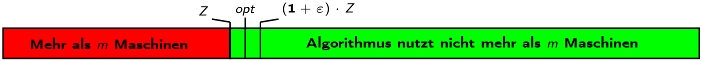
$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p'_i mit Makespan

$$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$

3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

- Es gibt $Z^* \leqslant opt$:
  das Bin Packing ist lösbar, falls $Z \geqslant Z^*$ gewählt wurde.



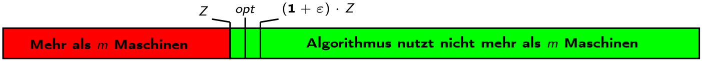
- Modifiziere Algorithmus so, dass er zu kleine Werte für Z erkennt.
- Falls Z aber zu groß ist, verlieren wir unseren Approximationsfaktor.

Das Orakel (Aufbau der Idee)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor$, $p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

1. Ein Orakel liefert Z, den Wert des optimalen Makespan.
2. Phase 1, die großen Jobs $G = \{i \in \{1, 2, \ldots, n\} \mid p_i > \varepsilon Z\}$:
   1. Skaliere die Größe der Jobs aus G: $p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$.
   2. Bestimme Schedule mit Jobgrößen p_i' mit Makespan
   $$Z' = \lfloor (1+\varepsilon)/\varepsilon^2 \rfloor.$$
3. Phase 2, die kleinen Jobs: $K = \{i \in \{1, 2, \ldots, n\} \mid p_i \leqslant \varepsilon Z\}$:
   1. Verteile die Jobs aus K nach der LL Heuristik.

- Es gibt $Z^* \leqslant opt$:
  das Bin Packing ist lösbar, falls $Z \geqslant Z^*$ gewählt wurde.



- Modifiziere Algorithmus so, dass er zu kleine Werte für Z erkennt.
- Falls Z aber zu groß ist, verlieren wir unseren Approximationsfaktor.
- Für eine $(1+\varepsilon)$-Approximation muss gelten: $Z^* \leqslant Z \leqslant opt$.

Das Orakel (Halbierungssuche)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Suche Z^* mit Binärsuche:

Das Orakel (Halbierungssuche)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Suche Z^* mit Binärsuche:
  - Starte mit $S = \sum_{i=1}^{n} p_i$ (Obere Schranke für Makespan).

Das Orakel (Halbierungssuche)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Suche Z^* mit Binärsuche:
  - Starte mit $S = \sum_{i=1}^{n} p_i$ (Obere Schranke für Makespan).
  - Damit ist der Wertebereich für die Binärsuche: $\{1, 2, \ldots, S\}$.

Das Orakel (Halbierungssuche)

$Z' = \lfloor (\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

- Suche Z^* mit Binärsuche:
  - Starte mit $S = \sum_{i=1}^{n} p_i$ (Obere Schranke für Makespan).
  - Damit ist der Wertebereich für die Binärsuche: $\{1, 2, \ldots, S\}$.
  - Anzahl der Aufrufe vom Algorithmus: $O(\log S)$.

Das Orakel (Halbierungssuche)

$Z' = \lfloor(\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}})\rfloor, \quad p'_i = \lceil p_i/(\varepsilon^{\mathbf{2}} Z)\rceil$

- Suche Z^* mit Binärsuche:
  - Starte mit $S = \sum_{i=1}^{n} p_i$ (Obere Schranke für Makespan).
  - Damit ist der Wertebereich für die Binärsuche: $\{1, 2, \ldots, S\}$.
  - Anzahl der Aufrufe vom Algorithmus: $O(\log S)$.
  - Sei N die Länge der Eingabe in Bits. Dann gilt: $\log S \leqslant N$.

Das Orakel (Halbierungssuche)

$Z' = \lfloor (\mathbf{1}+\varepsilon)/(\varepsilon^{\mathbf{2}}) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^{\mathbf{2}} Z) \rceil$

- Suche Z^* mit Binärsuche:
  - Starte mit $S = \sum_{i=1}^{n} p_i$ (Obere Schranke für Makespan).
  - Damit ist der Wertebereich für die Binärsuche: $\{1, 2, \ldots, S\}$.
  - Anzahl der Aufrufe vom Algorithmus: $O(\log S)$.
  - Sei N die Länge der Eingabe in Bits. Dann gilt: $\log S \leqslant N$.
  - Anzahl der Aufrufe vom Algorithmus: $O(N)$.

Das Orakel (Halbierungssuche)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Suche Z^* mit Binärsuche:
  - Starte mit $S = \sum_{i=1}^{n} p_i$ (Obere Schranke für Makespan).
  - Damit ist der Wertebereich für die Binärsuche: $\{1, 2, \ldots, S\}$.
  - Anzahl der Aufrufe vom Algorithmus: $O(\log S)$.
  - Sei N die Länge der Eingabe in Bits. Dann gilt: $\log S \leqslant N$.
  - Anzahl der Aufrufe vom Algorithmus: $O(N)$.
  - Gesamtlaufzeit: $O(N \cdot n^{\lceil 1/\varepsilon^2 \rceil})$

Das Orakel (Halbierungssuche)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Suche Z^* mit Binärsuche:
  - Starte mit $S = \sum_{i=1}^{n} p_i$ (Obere Schranke für Makespan).
  - Damit ist der Wertebereich für die Binärsuche: $\{1, 2, \ldots, S\}$.
  - Anzahl der Aufrufe vom Algorithmus: $O(\log S)$.
  - Sei N die Länge der Eingabe in Bits. Dann gilt: $\log S \leqslant N$.
  - Anzahl der Aufrufe vom Algorithmus: $O(N)$.
  - Gesamtlaufzeit: $O(N \cdot n^{\lceil 1/\varepsilon^2 \rceil})$

Das Orakel (Halbierungssuche)

$Z' = \lfloor (1+\varepsilon)/(\varepsilon^2) \rfloor, \quad p_i' = \lceil p_i/(\varepsilon^2 Z) \rceil$

- Suche Z^* mit Binärsuche:
  - Starte mit $S = \sum_{i=1}^n p_i$ (Obere Schranke für Makespan).
  - Damit ist der Wertebereich für die Binärsuche: $\{1, 2, \ldots, S\}$.
  - Anzahl der Aufrufe vom Algorithmus: $O(\log S)$.
  - Sei N die Länge der Eingabe in Bits. Dann gilt: $\log S \leqslant N$.
  - Anzahl der Aufrufe vom Algorithmus: $O(N)$.
  - Gesamtlaufzeit: $O(N \cdot n^{\lceil 1/\varepsilon^2 \rceil})$

Theorem

Es gibt ein PTAS für das Makespan-Scheduling-Problem auf identischen Maschinen.

Definitionen

Definition (Scheduling auf Maschinen mit Geschwindigkeiten)

Das Makespan Scheduling Problem auf Maschinen mit Geschwindigkeiten:

Definitionen

Definition (Scheduling auf Maschinen mit Geschwindigkeiten)

Das Makespan Scheduling Problem auf Maschinen mit Geschwindigkeiten:

- Gegeben:

Definitionen

Definition (Scheduling auf Maschinen mit Geschwindigkeiten)

Das Makespan Scheduling Problem auf Maschinen mit Geschwindigkeiten:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).

Definitionen

Definition (Scheduling auf Maschinen mit Geschwindigkeiten)

Das Makespan Scheduling Problem auf Maschinen mit Geschwindigkeiten:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $s_1, s_2, \ldots, s_m \in \mathbb{N}$ (d.h. m Maschinen mit Geschwindigkeit s_i).

Definitionen

Definition (Scheduling auf Maschinen mit Geschwindigkeiten)

Das Makespan Scheduling Problem auf Maschinen mit Geschwindigkeiten:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $s_1, s_2, \ldots, s_m \in \mathbb{N}$ (d.h. m Maschinen mit Geschwindigkeit s_i).
- Gesucht:

Definitionen

Definition (Scheduling auf Maschinen mit Geschwindigkeiten)

Das Makespan Scheduling Problem auf Maschinen mit Geschwindigkeiten:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $s_1, s_2, \ldots, s_m \in \mathbb{N}$ (d.h. m Maschinen mit Geschwindigkeit s_i).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$, (d.h. Zuweisung der Jobs auf Maschinen).

Definitionen

Definition (Scheduling auf Maschinen mit Geschwindigkeiten)

Das Makespan Scheduling Problem auf Maschinen mit Geschwindigkeiten:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $s_1, s_2, \ldots, s_m \in \mathbb{N}$ (d.h. m Maschinen mit Geschwindigkeit s_i).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$, (d.h. Zuweisung der Jobs auf Maschinen).
  - Mit $\max_{j \in \{1,2,\ldots,m\}} \sum_{i \in \{1,2,\ldots,n\} \wedge f(i)=j} \frac{p_i}{s_j}$ ist minimal.

Definitionen

Definition (Scheduling auf Maschinen mit Geschwindigkeiten)

Das Makespan Scheduling Problem auf Maschinen mit Geschwindigkeiten:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $s_1, s_2, \ldots, s_m \in \mathbb{N}$ (d.h. m Maschinen mit Geschwindigkeit s_i).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$, (d.h. Zuweisung der Jobs auf Maschinen).
  - Mit $\max_{j \in \{1,2,\ldots,m\}} \sum_{i \in \{1,2,\ldots,n\} \wedge f(i)=j} \frac{p_i}{s_j}$ ist minimal.

Definitionen

Definition (Scheduling auf Maschinen mit Geschwindigkeiten)

Das Makespan Scheduling Problem auf Maschinen mit Geschwindigkeiten:

- Gegeben:
  - $p_1, p_2, \ldots, p_n \in \mathbb{N}$ (d.h. n Jobs der Länge p_i).
  - $s_1, s_2, \ldots, s_m \in \mathbb{N}$ (d.h. m Maschinen mit Geschwindigkeit s_i).
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$, (d.h. Zuweisung der Jobs auf Maschinen).
  - Mit $\max_{j \in \{1,2,\ldots,m\}} \sum_{i \in \{1,2,\ldots,n\} \wedge f(i)=j} \frac{p_i}{s_j}$ ist minimal.

Bemerkung:

Das Scheduling auf Maschinen mit Geschwindigkeiten kann analog wie das Makespan-Scheduling-Problem auf identischen Maschinen gelöst werden.

Definitionen

Definition (Scheduling auf allgemeinen Maschinen)

Das Makespan Scheduling Problem auf allgemeinen Maschinen:

Definitionen

Definition (Scheduling auf allgemeinen Maschinen)

Das Makespan Scheduling Problem auf allgemeinen Maschinen:

- Gegeben:

Definitionen

Definition (Scheduling auf allgemeinen Maschinen)

Das Makespan Scheduling Problem auf allgemeinen Maschinen:

- Gegeben:
  - n Jobs und m Maschinen mit Laufzeiten:

Definitionen

Definition (Scheduling auf allgemeinen Maschinen)

Das Makespan Scheduling Problem auf allgemeinen Maschinen:

- Gegeben:
  - n Jobs und m Maschinen mit Laufzeiten:
  - $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.

Definitionen

Definition (Scheduling auf allgemeinen Maschinen)

Das Makespan Scheduling Problem auf allgemeinen Maschinen:

- Gegeben:
  - n Jobs und m Maschinen mit Laufzeiten:
  - $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
  - D.h. Job i hat auf Maschine j eine Laufzeit von $p_{i,j}$.

Definitionen

Definition (Scheduling auf allgemeinen Maschinen)

Das Makespan Scheduling Problem auf allgemeinen Maschinen:

- Gegeben:
  - n Jobs und m Maschinen mit Laufzeiten:
  - $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
  - D.h. Job i hat auf Maschine j eine Laufzeit von $p_{i,j}$.
- Gesucht:

Definitionen

Definition (Scheduling auf allgemeinen Maschinen)

Das Makespan Scheduling Problem auf allgemeinen Maschinen:

- Gegeben:
  - n Jobs und m Maschinen mit Laufzeiten:
  - $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
  - D.h. Job i hat auf Maschine j eine Laufzeit von $p_{i,j}$.
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$, (d.h. Zuweisung der Jobs auf Maschinen).

Definitionen

Definition (Scheduling auf allgemeinen Maschinen)

Das Makespan Scheduling Problem auf allgemeinen Maschinen:

- Gegeben:
  - n Jobs und m Maschinen mit Laufzeiten:
  - $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
  - D.h. Job i hat auf Maschine j eine Laufzeit von $p_{i,j}$.
- Gesucht:
  - $f : \{1, 2, \ldots, n\} \mapsto \{1, 2, \ldots, m\}$, (d.h. Zuweisung der Jobs auf Maschinen).
  - Mit $\max_{j \in \{1,2,\ldots,m\}} \sum_{i \in \{1,2,\ldots,n\} \wedge f(i)=j} p_{i,j}$ ist minimal.

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.
- So ein Gleichungssystem wird ILP (Integer Linear Programm) genannt.

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.
- So ein Gleichungssystem wird ILP (Integer Linear Programm) genannt.
  - Variablen $x_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.
- So ein Gleichungssystem wird ILP (Integer Linear Programm) genannt.
  - Variablen $x_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
  - Variable $t \in \mathbb{N}$.

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.
- So ein Gleichungssystem wird ILP (Integer Linear Programm) genannt.
  - Variablen $x_{i,j}$ für $i \in \{1,2,\ldots,n\}$ und $j \in \{1,2,\ldots,m\}$.
  - Variable $t \in \mathbb{N}$.
  - Notwendige Bedingungen (Nebenbedingungen)

$$
\begin{array}{ll}
\forall i \in \{1,2,\ldots,n\}: & \sum_{j\in\{1,2,\ldots,m\}} x_{i,j} \geqslant 1 \\
\forall j \in \{1,2,\ldots,m\}: & \sum_{i\in\{1,2,\ldots,n\}} x_{i,j} \cdot p_{i,j} \leqslant t \\
\forall i \in \{1,2,\ldots,n\}, j \in \{1,2,\ldots,m\}: & x_{i,j} \in \{0,1\}
\end{array}
$$

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.
- So ein Gleichungssystem wird ILP (Integer Linear Programm) genannt.
  - Variablen $x_{i,j}$ für $i \in \{1,2,\ldots,n\}$ und $j \in \{1,2,\ldots,m\}$.
  - Variable $t \in \mathbb{N}$.
  - Notwendige Bedingungen (Nebenbedingungen)

$$\begin{array}{ll}
\forall i \in \{1,2,\ldots,n\}: & \sum_{j\in\{1,2,\ldots,m\}} x_{i,j} \geqslant 1 \\
\forall j \in \{1,2,\ldots,m\}: & \sum_{i\in\{1,2,\ldots,n\}} x_{i,j}\cdot p_{i,j} \leqslant t \\
\forall i \in \{1,2,\ldots,n\}, j \in \{1,2,\ldots,m\}: & x_{i,j} \in \{0,1\}
\end{array}$$

  - Ziel: minimiere t.

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.
- So ein Gleichungssystem wird ILP (Integer Linear Programm) genannt.
  - Variablen $x_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
  - Variable $t \in \mathbb{N}$.
  - Notwendige Bedingungen (Nebenbedingungen)

$$\begin{array}{ll}
\forall i \in \{1, 2, \ldots, n\} : & \sum_{j \in \{1,2,\ldots,m\}} x_{i,j} \geqslant 1 \\
\forall j \in \{1, 2, \ldots, m\} : & \sum_{i \in \{1,2,\ldots,n\}} x_{i,j} \cdot p_{i,j} \leqslant t \\
\forall i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\} : & x_{i,j} \in \{0, 1\}
\end{array}$$

  - Ziel: minimiere t.
- So ein Gleichungssystem kann nicht in Polynomzeit gelöst werden, unter der Annahme $\mathcal{P} \neq \mathcal{NP}$.

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.
- So ein Gleichungssystem wird ILP (Integer Linear Programm) genannt.
  - Variablen $x_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
  - Variable $t \in \mathbb{N}$.
  - Notwendige Bedingungen (Nebenbedingungen)

$$\begin{array}{ll}
\forall i \in \{1, 2, \ldots, n\} : & \sum_{j \in \{1,2,\ldots,m\}} x_{i,j} \geqslant 1 \\
\forall j \in \{1, 2, \ldots, m\} : & \sum_{i \in \{1,2,\ldots,n\}} x_{i,j} \cdot p_{i,j} \leqslant t \\
\forall i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\} : & x_{i,j} \in \{0, 1\}
\end{array}$$

  - Ziel: minimiere t.
- So ein Gleichungssystem kann nicht in Polynomzeit gelöst werden, unter der Annahme $\mathcal{P} \neq \mathcal{NP}$.
- Aber ein relaxierte Variante ist in Polynomzeit lösbar:

$$\forall i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\} : 0 \leqslant x_{i,j} \leqslant 1.$$

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.
- So ein Gleichungssystem wird ILP (Integer Linear Programm) genannt.
  - Variablen $x_{i,j}$ für $i \in \{1,2,\ldots,n\}$ und $j \in \{1,2,\ldots,m\}$.
  - Variable $t \in \mathbb{N}$.
  - Notwendige Bedingungen (Nebenbedingungen)

$$\begin{array}{ll} \forall i \in \{1,2,\ldots,n\}: & \sum_{j\in\{1,2,\ldots,m\}} x_{i,j} \geqslant 1 \\ \forall j \in \{1,2,\ldots,m\}: & \sum_{i\in\{1,2,\ldots,n\}} x_{i,j} \cdot p_{i,j} \leqslant t \\ \forall i \in \{1,2,\ldots,n\}, j \in \{1,2,\ldots,m\}: & x_{i,j} \in \{0,1\} \end{array}$$

  - Ziel: minimiere t.
- So ein Gleichungssystem kann nicht in Polynomzeit gelöst werden, unter der Annahme $\mathcal{P} \neq \mathcal{NP}$.
- Aber ein relaxierte Variante ist in Polynomzeit lösbar:

$$\forall i \in \{1,2,\ldots,n\}, j \in \{1,2,\ldots,m\} : 0 \leqslant x_{i,j} \leqslant 1.$$

- Diese Variante kann aber schlecht zu einer guten Approximation führen.

ILP Formulierung

- Wir stellen das Problem als Gleichungssystem dar.
- Die Werte sind aber nicht aus $\mathbb{R}$ sondern aus $\mathbb{N}$.
- So ein Gleichungssystem wird ILP (Integer Linear Programm) genannt.
  - Variablen $x_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
  - Variable $t \in \mathbb{N}$.
  - Notwendige Bedingungen (Nebenbedingungen)

$$\begin{array}{ll} \forall i \in \{1, 2, \ldots, n\} : & \sum_{j \in \{1,2,\ldots,m\}} x_{i,j} \geqslant 1 \\ \forall j \in \{1, 2, \ldots, m\} : & \sum_{i \in \{1,2,\ldots,n\}} x_{i,j} \cdot p_{i,j} \leqslant t \\ \forall i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\} : & x_{i,j} \in \{0, 1\} \end{array}$$

  - Ziel: minimiere t.
- So ein Gleichungssystem kann nicht in Polynomzeit gelöst werden, unter der Annahme $\mathcal{P} \neq \mathcal{NP}$.
- Aber ein relaxierte Variante ist in Polynomzeit lösbar:

$$\forall i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, m\} : 0 \leqslant x_{i,j} \leqslant 1.$$

- Diese Variante kann aber schlecht zu einer guten Approximation führen.
- In der relaxierten Variante gilt: $n = 1 \wedge p_{1,j} = 1 \Longrightarrow t = 1/m$.

Alternative ILP Formulierung

- Falls der optimale Makespan Z bekannt ist (Idee mit dem Orakel), erhalten wir ein besseres Gleichungssystem.

Alternative ILP Formulierung

- Falls der optimale Makespan Z bekannt ist (Idee mit dem Orakel), erhalten wir ein besseres Gleichungssystem.
- Definiere $S_Z = \{(i,j) \in \{1,2,\ldots,n\} \times \{1,2,\ldots,m\} \mid p_{i,j} \leqslant Z\}$.

Alternative ILP Formulierung

- Falls der optimale Makespan Z bekannt ist (Idee mit dem Orakel), erhalten wir ein besseres Gleichungssystem.
- Definiere $S_Z = \{(i,j) \in \{1,2,\ldots,n\} \times \{1,2,\ldots,m\} \mid p_{i,j} \leqslant Z\}$.
- Damit erhalten wir folgendes relaxiertes Gleichungssystem:

Alternative ILP Formulierung

- Falls der optimale Makespan Z bekannt ist (Idee mit dem Orakel), erhalten wir ein besseres Gleichungssystem.
- Definiere $S_Z = \{(i,j) \in \{1,2,\ldots,n\} \times \{1,2,\ldots,m\} \mid p_{i,j} \leqslant Z\}$.
- Damit erhalten wir folgendes relaxiertes Gleichungssystem:
  - Variablen $x_{i,j}$ für $(i,j) \in S_Z$.

Alternative ILP Formulierung

- Falls der optimale Makespan Z bekannt ist (Idee mit dem Orakel), erhalten wir ein besseres Gleichungssystem.
- Definiere $S_Z = \{(i,j) \in \{1,2,\ldots,n\} \times \{1,2,\ldots,m\} \mid p_{i,j} \leqslant Z\}$.
- Damit erhalten wir folgendes relaxiertes Gleichungssystem:
  - Variablen $x_{i,j}$ für $(i,j) \in S_Z$.
  - Notwendige Bedingungen (Nebenbedingungen)

$$
\begin{array}{ll}
\forall i \in \{1,2,\ldots,n\}: & \sum_{j:(i,j)\in S_Z} x_{i,j} \geqslant 1 \\
\forall j \in \{1,2,\ldots,m\}: & \sum_{i:(i,j)\in S_Z} x_{i,j} \cdot p_{i,j} \leqslant Z \\
\forall (i,j) \in S_Z: & x_{i,j} \geqslant 0
\end{array}
$$

Alternative ILP Formulierung

- Falls der optimale Makespan Z bekannt ist (Idee mit dem Orakel), erhalten wir ein besseres Gleichungssystem.
- Definiere $S_Z = \{(i,j) \in \{1,2,\ldots,n\} \times \{1,2,\ldots,m\} \mid p_{i,j} \leqslant Z\}$.
- Damit erhalten wir folgendes relaxiertes Gleichungssystem:
  - Variablen $x_{i,j}$ für $(i,j) \in S_Z$.
  - Notwendige Bedingungen (Nebenbedingungen)

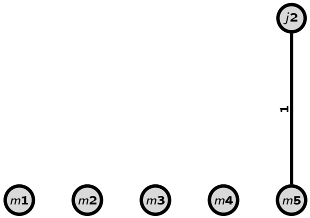
$$\begin{array}{ll} \forall i \in \{1,2,\ldots,n\}: & \sum_{j:(i,j)\in S_Z} x_{i,j} \geqslant 1 \\ \forall j \in \{1,2,\ldots,m\}: & \sum_{i:(i,j)\in S_Z} x_{i,j} \cdot p_{i,j} \leqslant Z \\ \forall (i,j) \in S_Z: & x_{i,j} \geqslant 0 \end{array}$$

  - Ziel: finde eine Lösung.

Situation

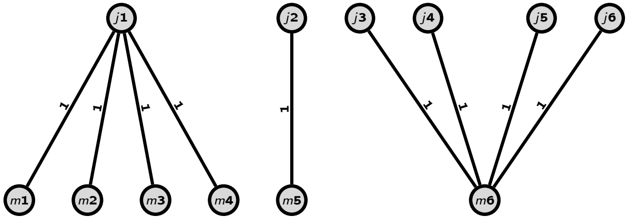
- Ein Job kann aufgespalten und mehreren Maschinen zugeteilt werden (siehe j_1).

Situation



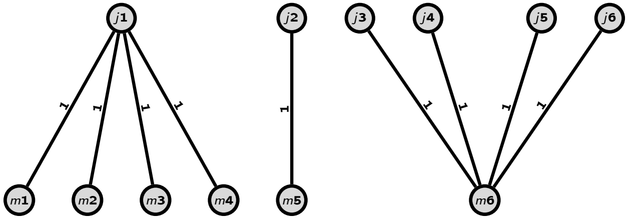
- Ein Job kann aufgespalten und mehreren Maschinen zugeteilt werden (siehe j_1).
- Ein Job kann vollständig auf eine Maschine zugeteilt werden (siehe j_2).

Situation



- Ein Job kann aufgespalten und mehreren Maschinen zugeteilt werden (siehe j_1).
- Ein Job kann vollständig auf eine Maschine zugeteilt werden (siehe j_2).
- Eine Maschine kann vollständige und aufgespaltene Jobs bekommen (siehe m_6).

Situation



- Ein Job kann aufgespalten und mehreren Maschinen zugeteilt werden (siehe j_1).
- Ein Job kann vollständig auf eine Maschine zugeteilt werden (siehe j_2).
- Eine Maschine kann vollständige und aufgespaltene Jobs bekommen (siehe m_6).
- Im Folgenden müssen diese "halben" Jobs eindeutig auf eine Maschine zugewiesen werden.

Algorithmus (Überblick)

1. Eingabe: $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.

Algorithmus (Überblick)

1. Eingabe: $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
2. Orakel bestimmt Z

Algorithmus (Überblick)

1. Eingabe: $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
2. Orakel bestimmt Z
3. Bestimme Alternative ILP Formulierung $ILP(Z)$.

Algorithmus (Überblick)

1. Eingabe: $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
2. Orakel bestimmt Z
3. Bestimme Alternative ILP Formulierung $ILP(Z)$.
4. Relaxiere $ILP(Z)$ zu $LP(Z)$.

Algorithmus (Überblick)

1. Eingabe: $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
2. Orakel bestimmt Z
3. Bestimme Alternative ILP Formulierung $ILP(Z)$.
4. Relaxiere $ILP(Z)$ zu $LP(Z)$.
5. Bestimme eine zulässige Basislösung B für $LP(Z)$.

Algorithmus (Überblick)

1. Eingabe: $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
2. Orakel bestimmt Z
3. Bestimme Alternative ILP Formulierung $ILP(Z)$.
4. Relaxiere $ILP(Z)$ zu $LP(Z)$.
5. Bestimme eine zulässige Basislösung B für $LP(Z)$.
6. Aus der Basislösung B bestimme durch geeignetes Auf- und Abrunden eine Lösung mit Approximationsfaktor 2.

Algorithmus (Überblick)

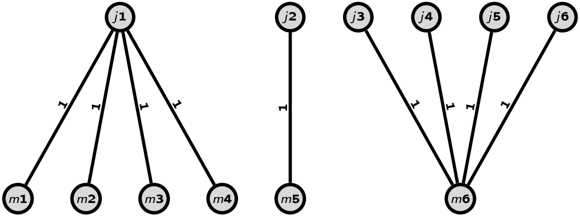
1. Eingabe: $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
2. Orakel bestimmt Z
3. Bestimme Alternative ILP Formulierung $ILP(Z)$.
4. Relaxiere $ILP(Z)$ zu $LP(Z)$.
5. Bestimme eine zulässige Basislösung B für $LP(Z)$.
6. Aus der Basislösung B bestimme durch geeignetes Auf- und Abrunden eine Lösung mit Approximationsfaktor 2.

Algorithmus (Überblick)

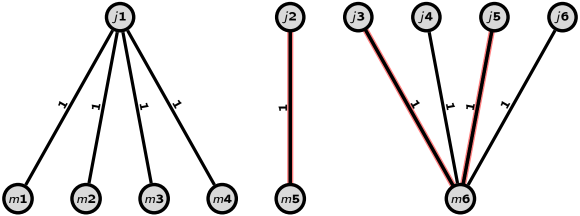
1. Eingabe: $p_{i,j}$ für $i \in \{1, 2, \ldots, n\}$ und $j \in \{1, 2, \ldots, m\}$.
2. Orakel bestimmt Z
3. Bestimme Alternative ILP Formulierung $ILP(Z)$.
4. Relaxiere $ILP(Z)$ zu $LP(Z)$.
5. Bestimme eine zulässige Basislösung B für $LP(Z)$.
6. Aus der Basislösung B bestimme durch geeignetes Auf- und Abrunden eine Lösung mit Approximationsfaktor 2.

- Das Bestimmen der Basislösung B für $LP(Z)$ erfolgt in Polymonzeit mit Hilfe der so genannten Elliqsoidmethode.

Situation

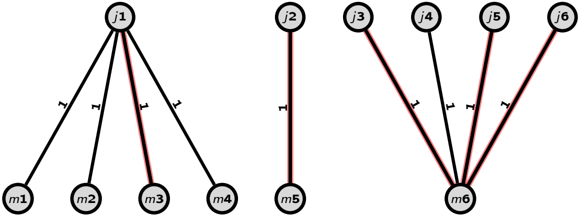


Situation



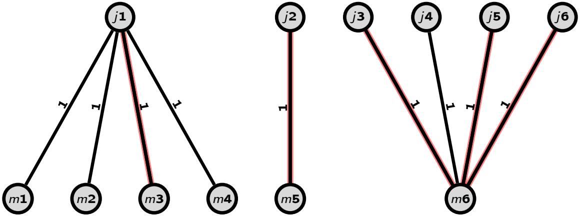
- Ein Job i mit $x_{i,j} = 1$ kann eindeutig auf m_j zugewiesen werden.

Situation



- Ein Job i mit $x_{i,j} = 1$ kann eindeutig auf m_j zugewiesen werden.
- Ein Job i mit $0 < x_{i,j} < 1$ kann ggf. auf m_j zugewiesen werden.

Situation



- Ein Job i mit $x_{i,j} = 1$ kann eindeutig auf m_j zugewiesen werden.
- Ein Job i mit $0 < x_{i,j} < 1$ kann ggf. auf m_j zugewiesen werden.
- Wird Job i mit $0 < x_{i,j} < 1$ auf m_j zugewiesen, so kann i nicht mehr an $m_{j'} \neq m_j$ mit $0 < x_{i,j'} < 1$ zugewiesen werden.

Anzahl der Variablen $x_{i,j}$ mit $x_{i,j} = 0$

Lemma

In der Basislösung B für $LP(Z)$ haben höchstens $n + m$ Variablen $x_{i,j}$ einen Wert $x_{i,j} > 0$.

Beweis:

Anzahl der Variablen $x_{i,j}$ mit $x_{i,j} = 0$

Lemma

In der Basislösung B für $LP(Z)$ haben höchstens $n + m$ Variablen $x_{i,j}$ einen Wert $x_{i,j} > 0$.

Beweis:

Anzahl der Variablen $x_{i,j}$ mit $x_{i,j} = 0$

Lemma

In der Basislösung B für $LP(Z)$ haben höchstens $n + m$ Variablen $x_{i,j}$ einen Wert $x_{i,j} > 0$.

Beweis:

- Sei D die Anzahl der Variablen in $LP(Z)$.

Anzahl der Variablen $x_{i,j}$ mit $x_{i,j} = 0$

Lemma

In der Basislösung B für $LP(Z)$ haben höchstens $n + m$ Variablen $x_{i,j}$ einen Wert $x_{i,j} > 0$.

Beweis:

- Sei D die Anzahl der Variablen in $LP(Z)$.
- Sei C die Anzahl der Nebenbedingungen (Ungleichungen) in $LP(Z)$.

Anzahl der Variablen $x_{i,j}$ mit $x_{i,j} = 0$

Lemma

In der Basislösung B für $LP(Z)$ haben höchstens $n + m$ Variablen $x_{i,j}$ einen Wert $x_{i,j} > 0$.

Beweis:

- Sei D die Anzahl der Variablen in $LP(Z)$.
- Sei C die Anzahl der Nebenbedingungen (Ungleichungen) in $LP(Z)$.
- Dann gilt:

$$D = |S_Z| \leqslant m \cdot n \text{ und } C = D + n + m.$$

Anzahl der Variablen $x_{i,j}$ mit $x_{i,j} = 0$

Lemma

In der Basislösung B für $LP(Z)$ haben höchstens $n + m$ Variablen $x_{i,j}$ einen Wert $x_{i,j} > 0$.

Beweis:

- Sei D die Anzahl der Variablen in $LP(Z)$.
- Sei C die Anzahl der Nebenbedingungen (Ungleichungen) in $LP(Z)$.
- Dann gilt:

$$D = |S_Z| \leqslant m \cdot n \text{ und } C = D + n + m.$$

- In der Basislösung B sind mindestens D der Nebenbedingungen exakt erfüllt.

Anzahl der Variablen $x_{i,j}$ mit $x_{i,j} = 0$

Lemma

In der Basislösung B für $LP(Z)$ haben höchstens $n+m$ Variablen $x_{i,j}$ einen Wert $x_{i,j} > 0$.

Beweis:

- Sei D die Anzahl der Variablen in $LP(Z)$.
- Sei C die Anzahl der Nebenbedingungen (Ungleichungen) in $LP(Z)$.
- Dann gilt:
$$D = |S_Z| \leqslant m \cdot n \text{ und } C = D + n + m.$$
- In der Basislösung B sind mindestens D der Nebenbedingungen exakt erfüllt.
- Also sind $C - D = n + m$ viele Nebenbedingungen nicht exakt erfüllt.

Anzahl der Variablen $x_{i,j}$ mit $x_{i,j} = 0$

Lemma

In der Basislösung B für $LP(Z)$ haben höchstens $n + m$ Variablen $x_{i,j}$ einen Wert $x_{i,j} > 0$.

Beweis:

- Sei D die Anzahl der Variablen in $LP(Z)$.
- Sei C die Anzahl der Nebenbedingungen (Ungleichungen) in $LP(Z)$.
- Dann gilt:
$$D = |S_Z| \leqslant m \cdot n \text{ und } C = D + n + m.$$
- In der Basislösung B sind mindestens D der Nebenbedingungen exakt erfüllt.
- Also sind $C - D = n + m$ viele Nebenbedingungen nicht exakt erfüllt.
- Also sind höchsten $n + m$ der Nebenbedingungen der Form $x_{i,j} \geqslant 0$ nicht exakt erfüllt.

Anzahl der Variablen $x_{i,j}$ mit $x_{i,j} = 0$

Lemma

In der Basislösung B für $LP(Z)$ haben höchstens $n + m$ Variablen $x_{i,j}$ einen Wert $x_{i,j} > 0$.

Beweis:

- Sei D die Anzahl der Variablen in $LP(Z)$.
- Sei C die Anzahl der Nebenbedingungen (Ungleichungen) in $LP(Z)$.
- Dann gilt:
$$D = |S_Z| \leqslant m \cdot n \text{ und } C = D + n + m.$$
- In der Basislösung B sind mindestens D der Nebenbedingungen exakt erfüllt.
- Also sind $C - D = n + m$ viele Nebenbedingungen nicht exakt erfüllt.
- Also sind höchsten $n + m$ der Nebenbedingungen der Form $x_{i,j} \geqslant 0$ nicht exakt erfüllt.
- Damit haben alle, bis auf höchstens $n + m$ Variablen, den Wert 0.

Allokationsgraph

Definition (Allokationsgraph)

Seien $x_{i,j}$ die Werte der Lösung für das $ILP(Z)$. Dann ist $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$, falls:

Allokationsgraph

Definition (Allokationsgraph)

Seien $x_{i,j}$ die Werte der Lösung für das $ILP(Z)$. Dann ist $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$, falls:

- $J = \{v_i \mid i \in \{1, 2, \ldots, n\}\}$,

Allokationsgraph

Definition (Allokationsgraph)

Seien $x_{i,j}$ die Werte der Lösung für das $ILP(Z)$. Dann ist $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$, falls:

- $J = \{v_i \mid i \in \{1, 2, \ldots, n\}\}$,
- $M = \{w_j \mid j \in \{1, 2, \ldots, m\}\}$ und

Allokationsgraph

Definition (Allokationsgraph)

Seien $x_{i,j}$ die Werte der Lösung für das $ILP(Z)$. Dann ist $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$, falls:

- $J = \{v_i \mid i \in \{1, 2, \ldots, n\}\}$,
- $M = \{w_j \mid j \in \{1, 2, \ldots, m\}\}$ und
- $E = \{(v_i, w_j) \mid x_{i,j} > 0\}$.

Allokationsgraph

Definition (Allokationsgraph)

Seien $x_{i,j}$ die Werte der Lösung für das $ILP(Z)$. Dann ist $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$, falls:

- $J = \{v_i \mid i \in \{1, 2, \ldots, n\}\}$,
- $M = \{w_j \mid j \in \{1, 2, \ldots, m\}\}$ und
- $E = \{(v_i, w_j) \mid x_{i,j} > 0\}$.

Allokationsgraph

Definition (Allokationsgraph)

Seien $x_{i,j}$ die Werte der Lösung für das $ILP(Z)$. Dann ist $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$, falls:

- $J = \{v_i \mid i \in \{1, 2, \ldots, n\}\}$,
- $M = \{w_j \mid j \in \{1, 2, \ldots, m\}\}$ und
- $E = \{(v_i, w_j) \mid x_{i,j} > 0\}$.

Lemma

Sei G der Allokationsgraph für $ILP(Z)$ und sei G zusammenhängend. Dann ist G ein Baum oder $G - e$ ist ein Baum für geeignetes e.

Beweis:

Allokationsgraph

Definition (Allokationsgraph)

Seien $x_{i,j}$ die Werte der Lösung für das $ILP(Z)$. Dann ist $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$, falls:

- $J = \{v_i \mid i \in \{1, 2, \ldots, n\}\}$,
- $M = \{w_j \mid j \in \{1, 2, \ldots, m\}\}$ und
- $E = \{(v_i, w_j) \mid x_{i,j} > 0\}$.

Lemma

Sei G der Allokationsgraph für $ILP(Z)$ und sei G zusammenhängend. Dann ist G ein Baum oder $G - e$ ist ein Baum für geeignetes e.

Beweis:

- G hat $m + n$ viele Knoten und höchstens $m + n$ viele Kanten.

Allokationsgraph

Definition (Allokationsgraph)

Seien $x_{i,j}$ die Werte der Lösung für das $ILP(Z)$. Dann ist $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$, falls:

- $J = \{v_i \mid i \in \{1, 2, \ldots, n\}\}$,
- $M = \{w_j \mid j \in \{1, 2, \ldots, m\}\}$ und
- $E = \{(v_i, w_j) \mid x_{i,j} > 0\}$.

Lemma

Sei G der Allokationsgraph für $ILP(Z)$ und sei G zusammenhängend. Dann ist G ein Baum oder $G - e$ ist ein Baum für geeignetes e.

Beweis:

- G hat $m + n$ viele Knoten und höchstens $m + n$ viele Kanten.
- Ein Baum mit k Knoten hat höchstens $k - 1$ viele Kanten.

Allokationsgraph

Definition (Allokationsgraph)

Seien $x_{i,j}$ die Werte der Lösung für das $ILP(Z)$. Dann ist $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$, falls:

- $J = \{v_i \mid i \in \{1, 2, \ldots, n\}\}$,
- $M = \{w_j \mid j \in \{1, 2, \ldots, m\}\}$ und
- $E = \{(v_i, w_j) \mid x_{i,j} > 0\}$.

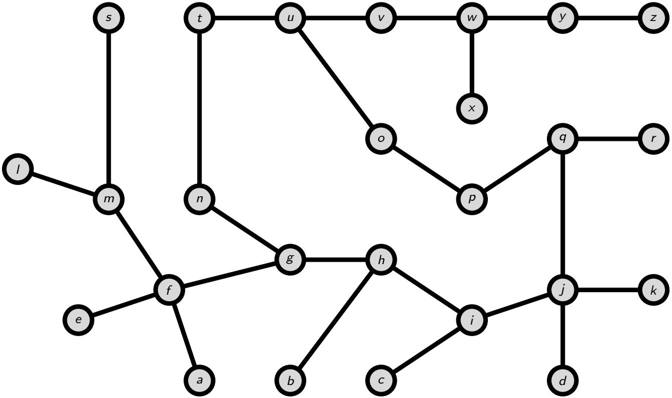
Lemma

Sei G der Allokationsgraph für $ILP(Z)$ und sei G zusammenhängend. Dann ist G ein Baum oder $G - e$ ist ein Baum für geeignetes e.

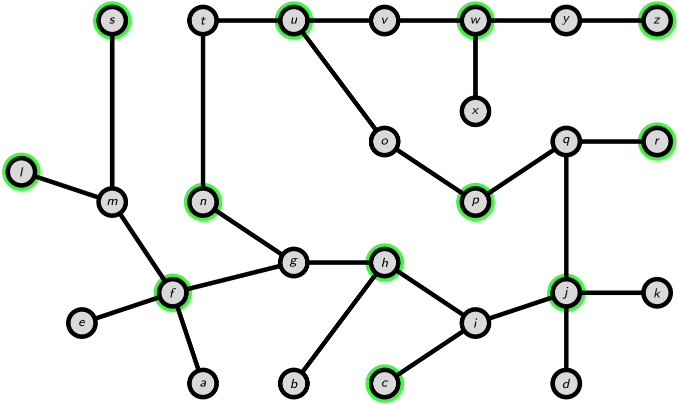
Beweis:

- G hat $m + n$ viele Knoten und höchstens $m + n$ viele Kanten.
- Ein Baum mit k Knoten hat höchstens $k - 1$ viele Kanten.
- D.h. G enthält höchstens einen Kreis.

Beispiel



Beispiel



Allokationsgraph

Lemma

Sei $G = (V, J, M)$ der Allokationsgraph für $ILP(Z)$ und sei G' Zusammenhangskomponente von G. Dann ist G' ein Baum oder $G' - e$ ist ein Baum für ein geeignetes e.

Beweis:

Allokationsgraph

Lemma

Sei $G = (V, J, M)$ der Allokationsgraph für ILP(Z) und sei G' Zusammenhangskomponente von G. Dann ist G' ein Baum oder $G' - e$ ist ein Baum für ein geeignetes e.

Beweis:

- Sei $G' = (V', W', E')$ eine beliebige Zusammenhangskomponente von G.

Allokationsgraph

Lemma

Sei $G = (V, J, M)$ der Allokationsgraph für $ILP(Z)$ und sei G' Zusammenhangskomponente von G. Dann ist G' ein Baum oder $G' - e$ ist ein Baum für ein geeignetes e.

Beweis:

- Sei $G' = (V', W', E')$ eine beliebige Zusammenhangskomponente von G.
- Das Tupel (V', W') definiert ein eingeschränktes Scheduling Problem.

Allokationsgraph

Lemma

Sei $G = (V, J, M)$ der Allokationsgraph für $ILP(Z)$ und sei G' Zusammenhangskomponente von G. Dann ist G' ein Baum oder $G' - e$ ist ein Baum für ein geeignetes e.

Beweis:

- Sei $G' = (V', W', E')$ eine beliebige Zusammenhangskomponente von G.
- Das Tupel (V', W') definiert ein eingeschränktes Scheduling Problem.
- Sei nun $LP'(Z)$ die Relaxierung dieses eingeschränkten Problems.

Allokationsgraph

Lemma

Sei $G = (V, J, M)$ der Allokationsgraph für $ILP(Z)$ und sei G' Zusammenhangskomponente von G. Dann ist G' ein Baum oder $G' - e$ ist ein Baum für ein geeignetes e.

Beweis:

- Sei $G' = (V', W', E')$ eine beliebige Zusammenhangskomponente von G.
- Das Tupel (V', W') definiert ein eingeschränktes Scheduling Problem.
- Sei nun $LP'(Z)$ die Relaxierung dieses eingeschränkten Problems.
- Wird Basislösung B auf die Variablen aus $V' \cup W'$ eingeschränkt, so erhalten wir eine Lösung für $LP'(Z)$.

Allokationsgraph

Lemma

Sei $G = (V, J, M)$ der Allokationsgraph für $ILP(Z)$ und sei G' Zusammenhangskomponente von G. Dann ist G' ein Baum oder $G' - e$ ist ein Baum für ein geeignetes e.

Beweis:

- Sei $G' = (V', W', E')$ eine beliebige Zusammenhangskomponente von G.
- Das Tupel (V', W') definiert ein eingeschränktes Scheduling Problem.
- Sei nun $LP'(Z)$ die Relaxierung dieses eingeschränkten Problems.
- Wird Basislösung B auf die Variablen aus $V' \cup W'$ eingeschränkt, so erhalten wir eine Lösung für $LP'(Z)$.
- Nach obigen Lemma gilt nun:

Allokationsgraph

Lemma

Sei $G = (V, J, M)$ der Allokationsgraph für $ILP(Z)$ und sei G' Zusammenhangskomponente von G. Dann ist G' ein Baum oder $G' - e$ ist ein Baum für ein geeignetes e.

Beweis:

- Sei $G' = (V', W', E')$ eine beliebige Zusammenhangskomponente von G.
- Das Tupel (V', W') definiert ein eingeschränktes Scheduling Problem.
- Sei nun $LP'(Z)$ die Relaxierung dieses eingeschränkten Problems.
- Wird Basislösung B auf die Variablen aus $V' \cup W'$ eingeschränkt, so erhalten wir eine Lösung für $LP'(Z)$.
- Nach obigen Lemma gilt nun:
  - G' ein Baum oder $G' - e$ ist eine Baum für geeignetes e.

Beispiel (Jobs sind die rot markierten Knoten)

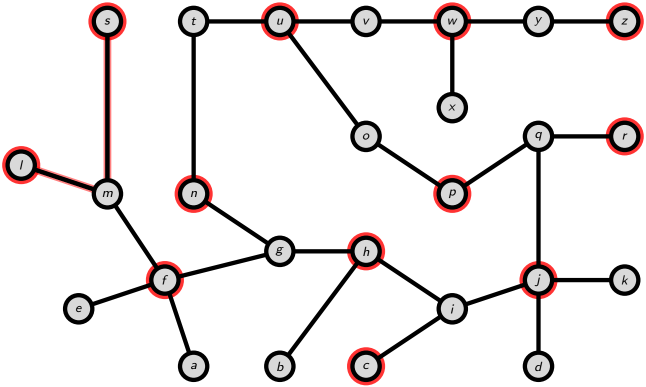
Beispiel (Jobs sind die rot markierten Knoten)

$l \to m$

Rot: $x_{i,j} = 1$

Beispiel (Jobs sind die rot markierten Knoten)

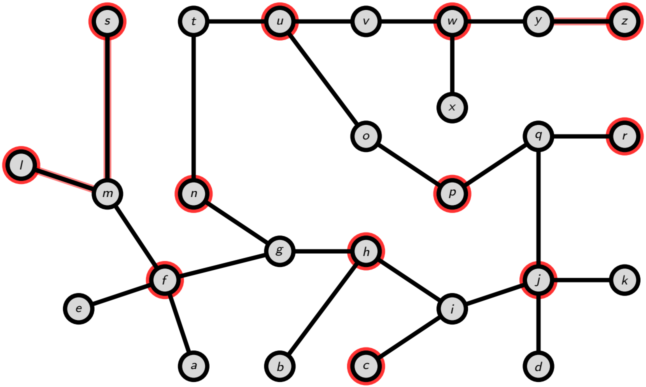
$l \rightarrow m$
$s \rightarrow m$



Rot: $x_{i,j} = 1$

Beispiel (Jobs sind die rot markierten Knoten)

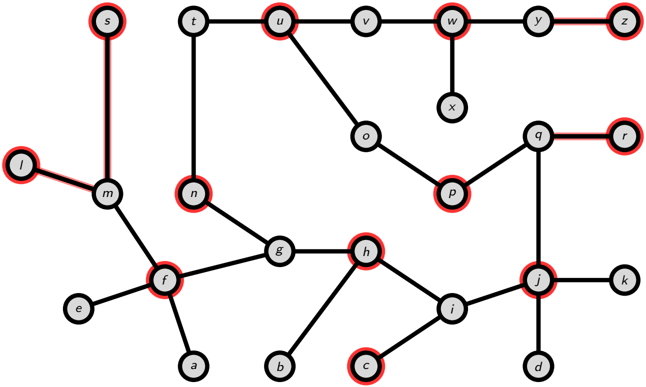
$l \rightarrow m$
$s \rightarrow m$
$z \rightarrow y$



Rot: $x_{i,j} = 1$

Beispiel (Jobs sind die rot markierten Knoten)

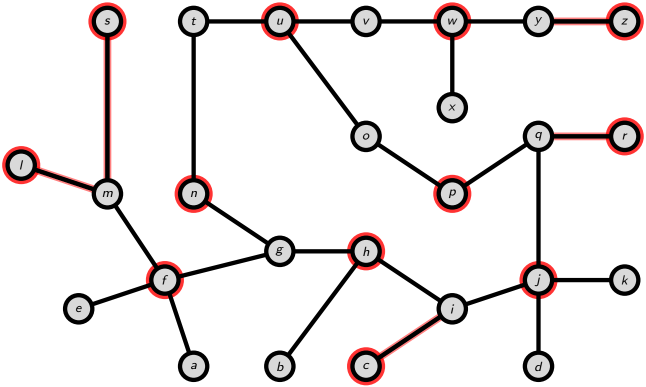
$l \to m$
$s \to m$
$z \to y$
$r \to q$



Rot: $x_{i,j} = 1$

Beispiel (Jobs sind die rot markierten Knoten)

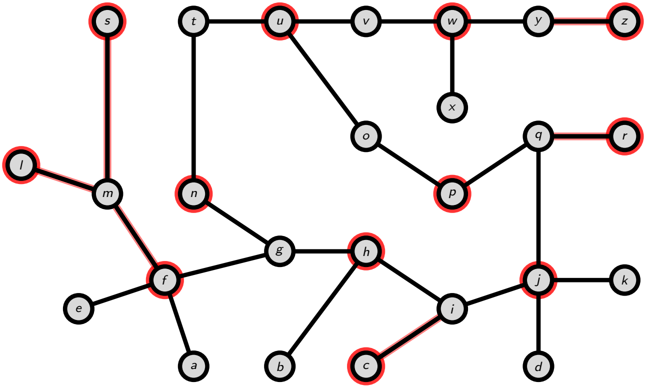
$l \to m$
$s \to m$
$z \to y$
$r \to q$
$c \to i$



Rot: $x_{i,j} = 1$

Beispiel (Jobs sind die rot markierten Knoten)

$l \to m$
$s \to m$
$z \to y$
$r \to q$
$c \to i$
$f \to m$

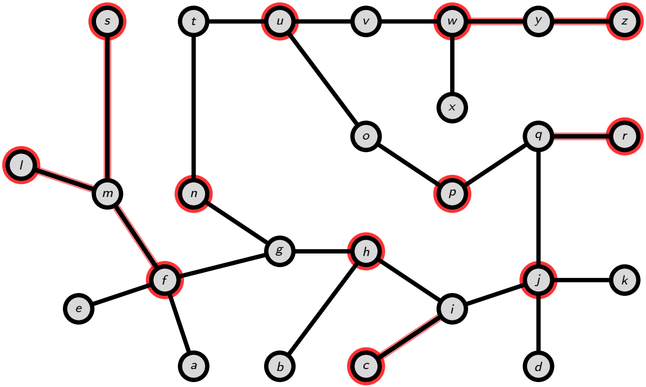


Rot: $x_{i,j} = 1$

Blau: $0 < x_{i,j} < 1$

Beispiel (Jobs sind die rot markierten Knoten)

$l \to m$
$s \to m$
$z \to y$
$r \to q$
$c \to i$
$f \to m$
$w \to y$

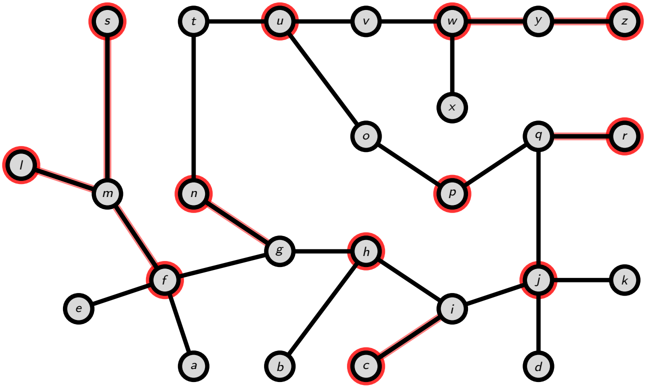


Rot: $x_{i,j} = 1$

Blau: $0 < x_{i,j} < 1$

Beispiel (Jobs sind die rot markierten Knoten)

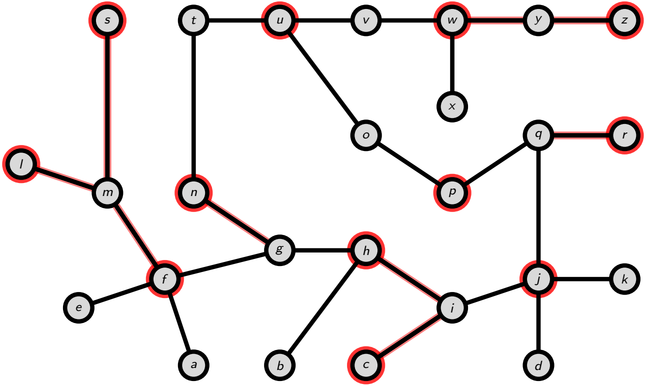
$l \to m$
$s \to m$
$z \to y$
$r \to q$
$c \to i$
$f \to m$
$w \to y$
$n \to g$



Rot: $x_{i,j} = 1$ Blau: $0 < x_{i,j} < 1$ Cyan: $0 < x_{i,j} < 1$

Beispiel (Jobs sind die rot markierten Knoten)

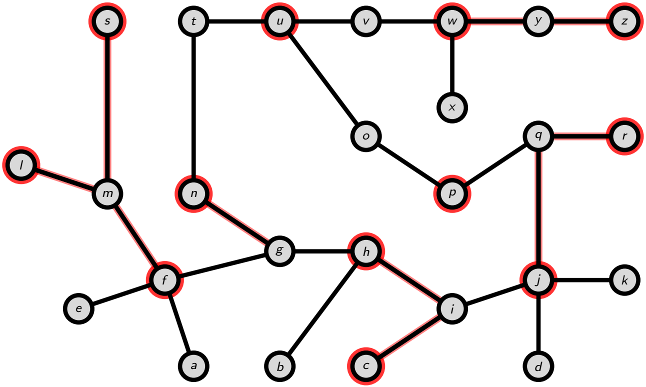
$l \to m$
$s \to m$
$z \to y$
$r \to q$
$c \to i$
$f \to m$
$w \to y$
$n \to g$
$h \to i$



Rot: $x_{i,j} = 1$ Blau: $0 < x_{i,j} < 1$ Cyan: $0 < x_{i,j} < 1$

Beispiel (Jobs sind die rot markierten Knoten)

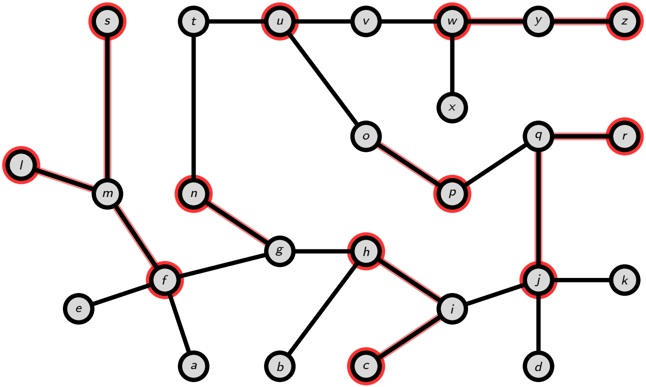
$l \rightarrow m$
$s \rightarrow m$
$z \rightarrow y$
$r \rightarrow q$
$c \rightarrow i$
$f \rightarrow m$
$w \rightarrow y$
$n \rightarrow g$
$h \rightarrow i$
$j \rightarrow q$



Rot: $x_{i,j} = 1$ Blau: $0 < x_{i,j} < 1$ Cyan: $0 < x_{i,j} < 1$

Beispiel (Jobs sind die rot markierten Knoten)

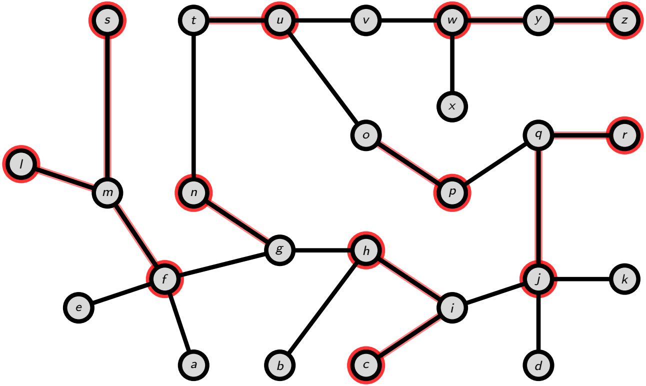
$l \to m$
$s \to m$
$z \to y$
$r \to q$
$c \to i$
$f \to m$
$w \to y$
$n \to g$
$h \to i$
$j \to q$
$p \to o$



Rot: $x_{i,j} = 1$ Blau: $0 < x_{i,j} < 1$ Cyan: $0 < x_{i,j} < 1$

Beispiel (Jobs sind die rot markierten Knoten)

$l \rightarrow m$
$s \rightarrow m$
$z \rightarrow y$
$r \rightarrow q$
$c \rightarrow i$
$f \rightarrow m$
$w \rightarrow y$
$n \rightarrow g$
$h \rightarrow i$
$j \rightarrow q$
$p \rightarrow o$
$u \rightarrow t$



Rot: $x_{i,j} = 1$ Blau: $0 < x_{i,j} < 1$ Cyan: $0 < x_{i,j} < 1$

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.
- Nachdem alle ungeteilten Jobs zugewiesen worden sind, hat der verbleibende Graph G keine Blätter aus J.

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.
- Nachdem alle ungeteilten Jobs zugewiesen worden sind, hat der verbleibende Graph G keine Blätter aus J.
- Betrachte geteilte Jobs, d.h. $0 < x_{i,j} < 1$:

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.
- Nachdem alle ungeteilten Jobs zugewiesen worden sind, hat der verbleibende Graph G keine Blätter aus J.
- Betrachte geteilte Jobs, d.h. $0 < x_{i,j} < 1$:
  - Berechne einseitig perfektes Matching für M.

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.
- Nachdem alle ungeteilten Jobs zugewiesen worden sind, hat der verbleibende Graph G keine Blätter aus J.
- Betrachte geteilte Jobs, d.h. $0 < x_{i,j} < 1$:
  - Berechne einseitig perfektes Matching für M.
  - Falls w_j ein isolierter Knoten ist, entferne ihn aus G.

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.
- Nachdem alle ungeteilten Jobs zugewiesen worden sind, hat der verbleibende Graph G keine Blätter aus J.
- Betrachte geteilte Jobs, d.h. $0 < x_{i,j} < 1$:
  - Berechne einseitig perfektes Matching für M.
  - Falls w_j ein isolierter Knoten ist, entferne ihn aus G.
  - Falls $w_j \in M$ ein Blatt (Knoten vom Grad 1) ist und v_i der eindeutige Nachbar von w_j, dann setze:

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.
- Nachdem alle ungeteilten Jobs zugewiesen worden sind, hat der verbleibende Graph G keine Blätter aus J.
- Betrachte geteilte Jobs, d.h. $0 < x_{i,j} < 1$:
  - Berechne einseitig perfektes Matching für M.
  - Falls w_j ein isolierter Knoten ist, entferne ihn aus G.
  - Falls $w_j \in M$ ein Blatt (Knoten vom Grad 1) ist und v_i der eindeutige Nachbar von w_j, dann setze:
    - Setze $f(i) = j$ und

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.
- Nachdem alle ungeteilten Jobs zugewiesen worden sind, hat der verbleibende Graph G keine Blätter aus J.
- Betrachte geteilte Jobs, d.h. $0 < x_{i,j} < 1$:
  - Berechne einseitig perfektes Matching für M.
  - Falls w_j ein isolierter Knoten ist, entferne ihn aus G.
  - Falls $w_j \in M$ ein Blatt (Knoten vom Grad 1) ist und v_i der eindeutige Nachbar von w_j, dann setze:
    - Setze $f(i) = j$ und
    - setze $G = G|(J \setminus \{v_i\}) \cup (M \setminus \{w_j\})$.

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.
- Nachdem alle ungeteilten Jobs zugewiesen worden sind, hat der verbleibende Graph G keine Blätter aus J.
- Betrachte geteilte Jobs, d.h. $0 < x_{i,j} < 1$:
  - Berechne einseitig perfektes Matching für M.
  - Falls w_j ein isolierter Knoten ist, entferne ihn aus G.
  - Falls $w_j \in M$ ein Blatt (Knoten vom Grad 1) ist und v_i der eindeutige Nachbar von w_j, dann setze:
    - Setze $f(i) = j$ und
    - setze $G = G|(J \setminus \{v_i\}) \cup (M \setminus \{w_j\})$.
  - Falls $G = (J, M, E)$ einen Kreis C von Knoten vom Grad 2 enthält, dann bestimme perfektes Matching M auf C.

Runden mit Hilfe des Allokationsgraphen

- Gegeben sei $G = (J, M, E)$ der Allokationsgraph für $ILP(Z)$ und Variablenwerte $x_{i,j}$ aus der Basislösung.
- Betrachte ungeteilte Jobs, d.h. $x_{i,j} = 1$:
  - Setze $f(i) = j$ und
  - setze $G = G|(J \setminus \{v_i\}) \cup M$.
- Nachdem alle ungeteilten Jobs zugewiesen worden sind, hat der verbleibende Graph G keine Blätter aus J.
- Betrachte geteilte Jobs, d.h. $0 < x_{i,j} < 1$:
  - Berechne einseitig perfektes Matching für M.
  - Falls w_j ein isolierter Knoten ist, entferne ihn aus G.
  - Falls $w_j \in M$ ein Blatt (Knoten vom Grad 1) ist und v_i der eindeutige Nachbar von w_j, dann setze:
    - Setze $f(i) = j$ und
    - setze $G = G|(J \setminus \{v_i\}) \cup (M \setminus \{w_j\})$.
  - Falls $G = (J, M, E)$ einen Kreis C von Knoten vom Grad 2 enthält, dann bestimme perfektes Matching M auf C.
  - Setze für $(v_i, w_j) \in M : f(i) = j$ und entferne C aus G.

Approximation

Theorem

Der obige Algorithmus bestimmt eine 2-Approximation für das allgemeine Makespan Scheduling Problem.

Beweis:

Approximation

Theorem

Der obige Algorithmus bestimmt eine 2-Approximation für das allgemeine Makespan Scheduling Problem.

Beweis:

Approximation

Theorem

Der obige Algorithmus bestimmt eine 2-Approximation für das allgemeine Makespan Scheduling Problem.

Beweis:

- Die Verteilung der ungeteilten Jobs erzeugt auf jeder Maschine eine Last von höchstens Z.

Approximation

Theorem

Der obige Algorithmus bestimmt eine 2-Approximation für das allgemeine Makespan Scheduling Problem.

Beweis:

- Die Verteilung der ungeteilten Jobs erzeugt auf jeder Maschine eine Last von höchstens Z.
- Für die geteilten Jobs gilt:

Approximation

Theorem

Der obige Algorithmus bestimmt eine 2-Approximation für das allgemeine Makespan Scheduling Problem.

Beweis:

- Die Verteilung der ungeteilten Jobs erzeugt auf jeder Maschine eine Last von höchstens Z.
- Für die geteilten Jobs gilt:
  - $x_{i,j} > 0$ und damit $(i,j) \in S_Z$.

Approximation

Theorem

Der obige Algorithmus bestimmt eine 2-Approximation für das allgemeine Makespan Scheduling Problem.

Beweis:

- Die Verteilung der ungeteilten Jobs erzeugt auf jeder Maschine eine Last von höchstens Z.
- Für die geteilten Jobs gilt:
  - $x_{i,j} > 0$ und damit $(i,j) \in S_Z$.
  - Damit gilt: $p_{i,j} \leqslant Z$.

Approximation

Theorem

Der obige Algorithmus bestimmt eine 2-Approximation für das allgemeine Makespan Scheduling Problem.

Beweis:

- Die Verteilung der ungeteilten Jobs erzeugt auf jeder Maschine eine Last von höchstens Z.
- Für die geteilten Jobs gilt:
  - $x_{i,j} > 0$ und damit $(i,j) \in S_Z$.
  - Damit gilt: $p_{i,j} \leqslant Z$.
  - Jede Maschine erhält höchstens einen ungeteilten Job zugewiesen.

Approximation

Theorem

Der obige Algorithmus bestimmt eine 2-Approximation für das allgemeine Makespan Scheduling Problem.

Beweis:

- Die Verteilung der ungeteilten Jobs erzeugt auf jeder Maschine eine Last von höchstens Z.
- Für die geteilten Jobs gilt:
  - $x_{i,j} > 0$ und damit $(i,j) \in S_Z$.
  - Damit gilt: $p_{i,j} \leqslant Z$.
  - Jede Maschine erhält höchstens einen ungeteilten Job zugewiesen.
  - Damit ergibt sich eine maximale Lastzunahme von Z für jede Maschine.

Nichtapproximierbarkeit

- Die folgenden Probleme sind MAX-SNP schwer:

Nichtapproximierbarkeit

- Die folgenden Probleme sind MAX-SNP schwer:
  - Vertex Cover

Nichtapproximierbarkeit

- Die folgenden Probleme sind MAX-SNP schwer:
  - Vertex Cover
  - Zentrumsproblem

Nichtapproximierbarkeit

- Die folgenden Probleme sind MAX-SNP schwer:
  - Vertex Cover
  - Zentrumsproblem
  - Δ-TSP

Nichtapproximierbarkeit

- Die folgenden Probleme sind MAX-SNP schwer:
  - Vertex Cover
  - Zentrumsproblem
  - Δ-TSP
  - Steiner-Baum

Nichtapproximierbarkeit

- Die folgenden Probleme sind MAX-SNP schwer:
  - Vertex Cover
  - Zentrumsproblem
  - Δ-TSP
  - Steiner-Baum
- D.h. für diese gibt es kein PTAS.

Fragen

- Wie kann das Problem Set-Cover approximiert werden? Welche untere Schranken sind dazu bekannt?

Fragen

- Wie kann das Problem Set-Cover approximiert werden? Welche untere Schranken sind dazu bekannt?
- Wie arbeitet die LL Heuristik?

Fragen

- Wie kann das Problem Set-Cover approximiert werden? Welche untere Schranken sind dazu bekannt?
- Wie arbeitet die LL Heuristik?
- Wie arbeitet die LPT Heuristik?

Fragen

- Wie kann das Problem Set-Cover approximiert werden? Welche untere Schranken sind dazu bekannt?
- Wie arbeitet die LL Heuristik?
- Wie arbeitet die LPT Heuristik?
- Wie ist die Güte der LL Heuristik? Wie ist der Beweis?

Fragen

- Wie kann das Problem Set-Cover approximiert werden? Welche untere Schranken sind dazu bekannt?
- Wie arbeitet die LL Heuristik?
- Wie arbeitet die LPT Heuristik?
- Wie ist die Güte der LL Heuristik? Wie ist der Beweis?
- Wie ist die Güte der LPT Heuristik? Wie ist der Beweis?

Fragen

- Wie kann das Problem Set-Cover approximiert werden? Welche untere Schranken sind dazu bekannt?
- Wie arbeitet die LL Heuristik?
- Wie arbeitet die LPT Heuristik?
- Wie ist die Güte der LL Heuristik? Wie ist der Beweis?
- Wie ist die Güte der LPT Heuristik? Wie ist der Beweis?
- Wie arbeitet das Approximationsschema für Makespan Scheduling? Wie ist die Beweisidee?

Fragen

- Wie kann das Problem Set-Cover approximiert werden? Welche untere Schranken sind dazu bekannt?
- Wie arbeitet die LL Heuristik?
- Wie arbeitet die LPT Heuristik?
- Wie ist die Güte der LL Heuristik? Wie ist der Beweis?
- Wie ist die Güte der LPT Heuristik? Wie ist der Beweis?
- Wie arbeitet das Approximationsschema für Makespan Scheduling? Wie ist die Beweisidee?
- Wie kann des Makespan Problem auf allgemeinen Maschinen approximiert werden? Wie ist die Beweisidee?