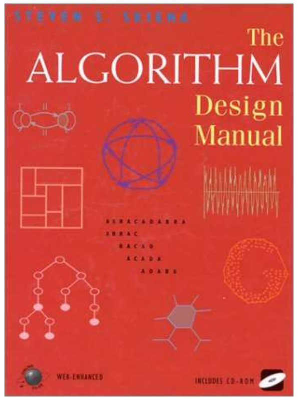
Literatur

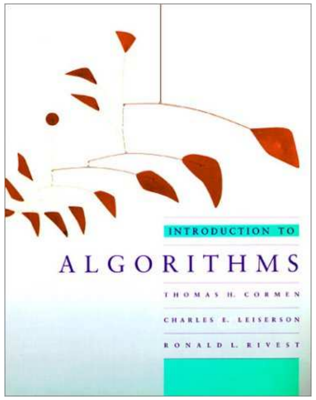


Sehr empfehlenswertes Buch. Es enthält einen Überblick über die wichtigsten Entwurfstechniken für Algorithmen und einen sehr vollständigen Überblick über bekannte Algorithmen mit Verweisen.

Steven S. Skiena: *The Algorithm Design Manual.* Springer Verlag.

Preis: US $69.95, Hardcover mit CD-ROM.

Literatur



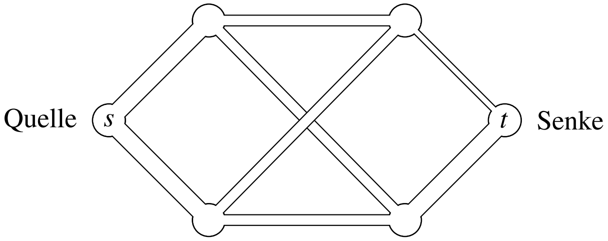
Sehr umfangreiches, modernes Buch.
Sehr gut geschrieben.
Enthält alle wichtigen Algorithmen.
Nur ein Buch ⇒ dieses!

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest: *Introduction To Algorithms.* MIT Press.

Preis: US $69.95, Hardcover, 1028 Seiten!

Netzwerkfluß

Gegeben ist ein System von Wasserrohren:

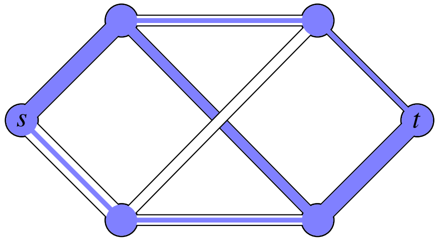


Die Kapazität jedes Rohres ist 3, 5 oder 8 l/s.

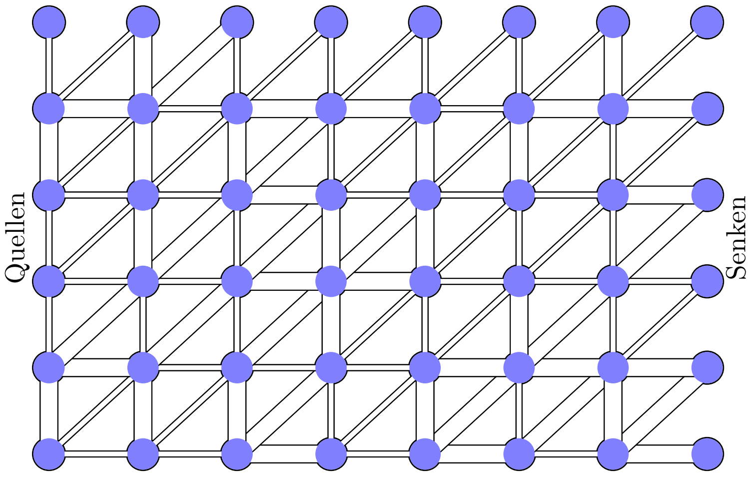
Frage: Wieviel Wasser kann von der Quelle zur Senke fließen?

Netzwerkfluß

Antwort: Maximal 11 l/s sind möglich.

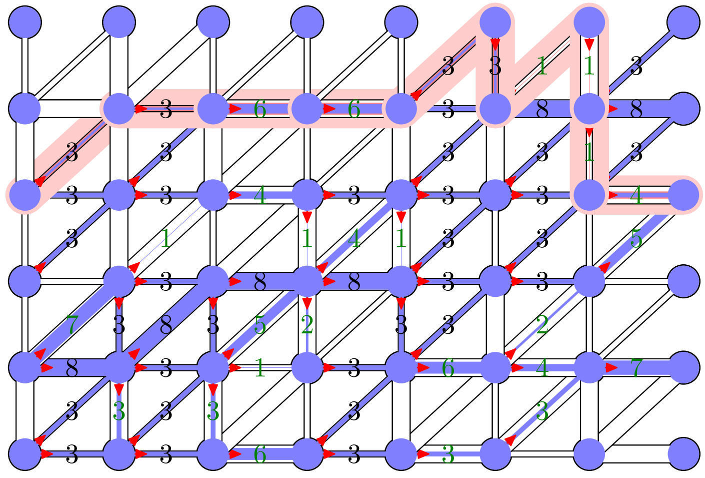


Netzwerkfluß – Aufgabe



Die Kapazitäten sind 3 und 8.

Netzwerkfluß – Lösung



Der maximale Fluß beträgt 30.

s-t-Netzwerke

Ein *s-t-Netzwerk* (flow network) ist ein gerichteter Graph $G = (V, E)$, wobei

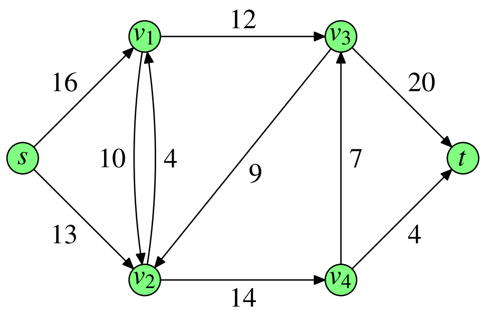
1. jede Kante $(u, v) \in E$ eine *Kapazität* $c(u, v) \geq 0$ hat,
2. es eine *Quelle* $s \in V$ und eine *Senke* $t \in V$ gibt.

Es ist bequem, anzunehmen daß jeder Knoten auf einem Pfad von s nach t liegt.

Falls $(u, v) \notin E$ setzen wir $c(u, v) = 0$.

Es kann Kanten (u, v) und (v, u) mit verschiedener Kapazität geben.

Beispiel eines s-t-Netzwerks



Die Kanten sind mit den Kapazitäten $c(u, v)$ beschriftet.

Flüsse

Ein *Fluß* ist eine Funktion $f\colon V \times V \to \mathbf{R}$, die Paare von Knoten auf reelle Zahlen abbildet und diese Bedingungen erfüllt:

- *Zulässigkeit:* Für $u, v \in V$ gilt $f(u,v) \le c(u,v)$.
- *Symmetrie:* Für $u, v \in V$ gilt $f(u,v) = -f(v,u)$.
- *Flußerhaltung:* Für $u \in V - \{s,t\}$ gilt $\sum_{v \in V} f(u,v) = 0$.

Der *Betrag* $|f|$ eines Flusses ist definiert als $|f| = \sum_{u \in V} f(s,u)$.

Dies ist gerade der Gesamtfluß aus der Quelle heraus.

Maximale Flüsse

Das Problem des *maximalen Flusses*:

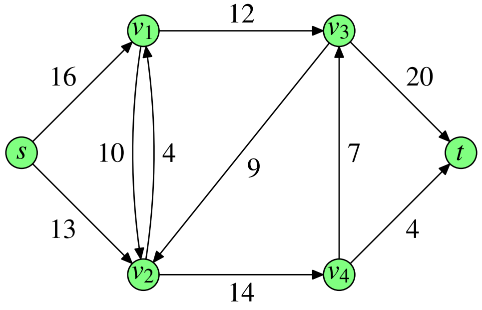
Gegeben:

Ein s-t-Netzwerk.

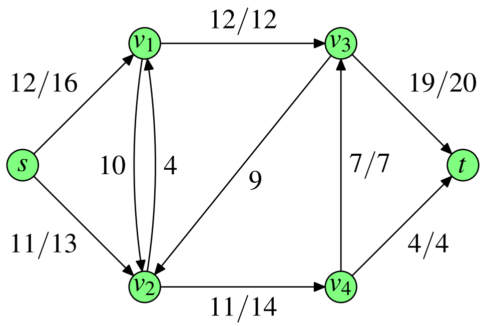
Gesucht:

Ein Fluß mit maximalem Betrag.

Was ist der maximale Fluß?

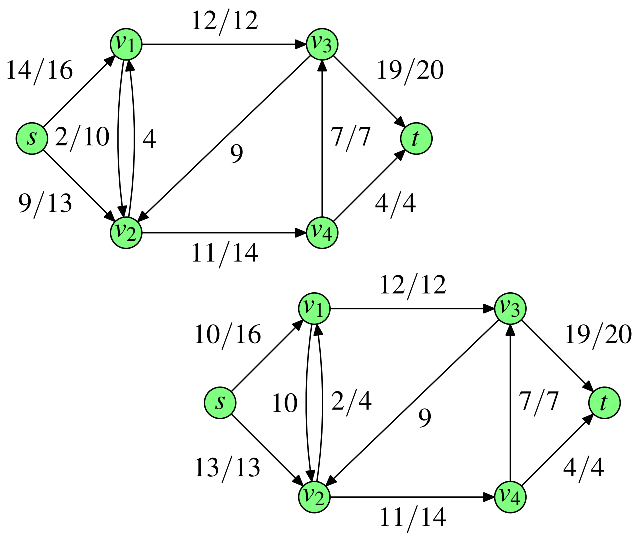


Der maximale Fluß ist 23

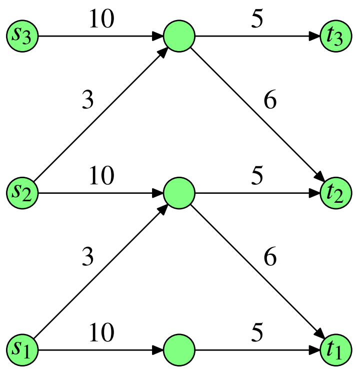


Die Kanten sind mit $c(u,v)$ beschriftet oder mit $f(u,v)/c(u,v)$, falls $f(u,v) > 0$.

Andere optimale Lösungen

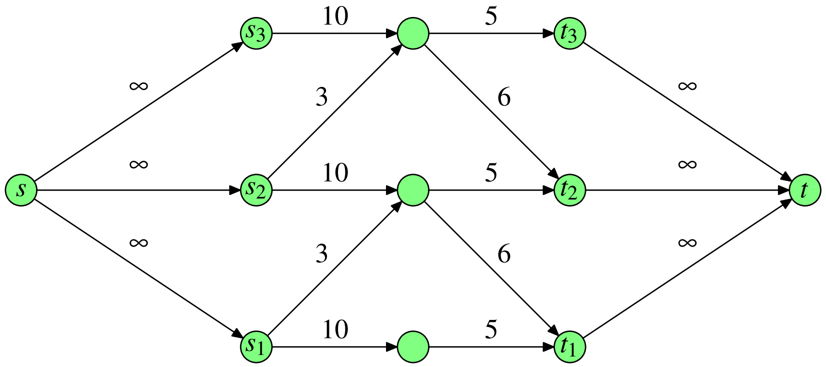


Mehrfache Quellen oder Senken



Mehrfache Quellen oder Senken sind eine **Verallgemeinerung** des Problem des maximalen Flusses.

Mehrfache Quellen oder Senken



→ Neue „Superquelle“ und „Supersenke“ hinzufügen

Existenz des maximalen Flusses

Existiert stets der maximale Fluß

$$\max\left\{ |f| \;\middle|\; f \text{ ist ein } s\text{-}t\text{-Fluß in } G \right\}?$$

Ja, denn die Menge aller Flüsse ist abgeschlossen im $\mathbf{R}^m$ und sie ist nicht leer.

Die stetige Funktion, die einen Fluß auf ihren Betrag abbildet, hat daher ein Maximum:

$$|f| = \sum_{u \in V} f(s,u) \text{ ist stetig!}$$

Einige Notationen

Es ist bequem einige Abkürzungen zu verwenden:

- $f(X,Y) = \sum_{x \in X} \sum_{y \in Y} f(x,y)$ für $X, Y \subseteq V$
- $f(x,Y) = \sum_{y \in Y} f(x,y)$ für $Y \subseteq V$
- $f(X,y) = \sum_{x \in X} f(x,y)$ für $X \subseteq V$
- $X - y$ statt $X - \{y\}$

Lemma A

Falls f ein s-t-Fluß für $G = (V, E)$ ist, dann gilt:

1. $f(X,X) = 0$ für $X \subseteq V$
2. $f(X,Y) = -f(Y,X)$ für $X, Y \subseteq V$
3. $f(X \cup Y, Z) = f(X,Z) + f(Y,Z)$ für $X, Y, Z \subseteq V$ mit $X \cap Y = \emptyset$
4. $f(Z, X \cup Y) = f(Z,X) + f(Z,Y)$ für $X, Y, Z \subseteq V$ mit $X \cap Y = \emptyset$

Dieses Lemma ist sehr nützlich, um wichtige Eigenschaften über Flüsse abzuleiten.

Lemma A

Um Lemma A zu beweisen, dürfen wir nur die Eigenschaften eines s-t-Flusses verwenden, also Zulässigkeit, Symmetrie und Flußerhaltung.

Beweis für $f(X,X) = 0$:

$$\begin{aligned} f(X,X) &= \frac{1}{2}\Big(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1,x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1,x_2) \Big) \\ &= \frac{1}{2}\Big(\sum_{x_1 \in X} \sum_{x_2 \in X} f(x_1,x_2) + \sum_{x_1 \in X} \sum_{x_2 \in X} f(x_2,x_1) \Big) \\ &= \frac{1}{2} \sum_{x_1 \in X} \sum_{x_2 \in X} \Big(f(x_1,x_2) + f(x_2,x_1) \Big) = 0 \end{aligned}$$

Hier genügt die Symmetrie allein! (Rest als Übungsaufgabe.)

Anwendung des Lemmas

Der Fluß in die Senke sollte intuitiv dem Fluß aus der Quelle entsprechen:

$$f(s,V) = f(V,t)$$

Beweis mit Lemma A:

$$\begin{aligned} f(s,V) &= f(V,V) - f(V-s,V) \\ &= -f(V-s,V) \\ &= f(V,V-s) \\ &= f(V,t) + f(V,V-s-t) \\ &= f(V,t) \text{ wegen Flußerhaltung} \end{aligned}$$

Residualnetzwerke

„Netzwerk minus Fluß = Residualnetzwerk“

Definition:

Gegeben ist ein Netzwerk $G = (V, E)$ und ein Fluß f. Das *Residualnetzwerk* $G_f = (V, E_f)$ zu G und f ist definiert vermöge

$$E_f = \{ (u, v) \in V \times V \mid c_f(u, v) > 0 \},$$

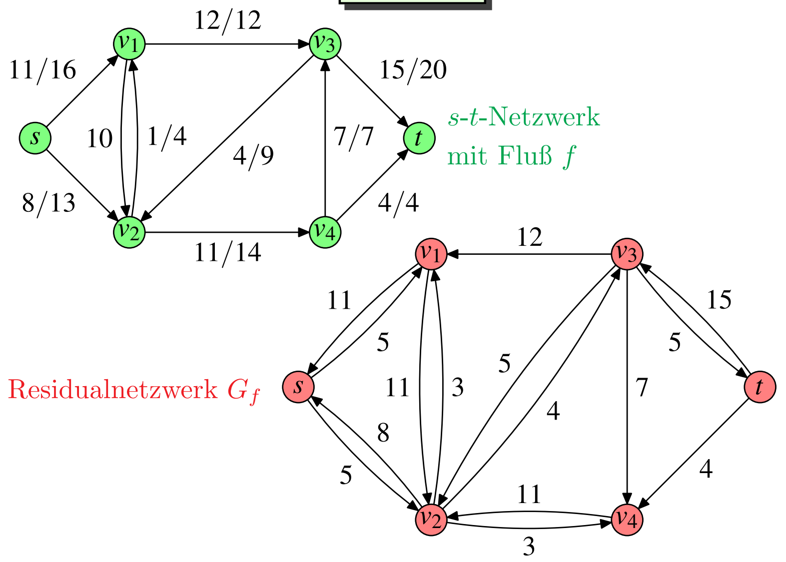
wobei

$$c_f(u, v) = c(u, v) - f(u, v).$$

c_f ist die *Restkapazität*.

Das s-t-Netzwerk G_f hat die Kapazitäten c_f.

Beispiel

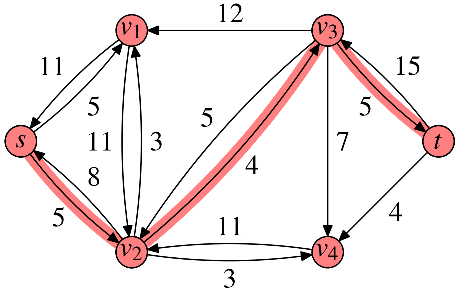


Augmentierende Pfade

Ein s-t-Pfad p in G_f heißt *augmentierender Pfad*.

$c_f(p) = \min\{ c_f(u, v) \mid (u, v) \text{ ist auf } p \}$ heißt *Restkapazität* von p.

Beispiel:



Die Restkapazität dieses Pfades ist 4.

Die Ford–Fulkerson–Methode

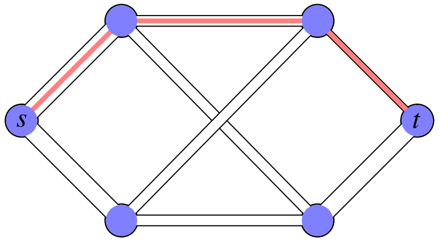
Initialisiere Fluß f zu 0
while es gibt einen augmentierenden Pfad p
  do augmentiere f entlang p
return f

$$f_p(u, v) = \begin{cases} c_f(p) & \text{falls } (u, v) \text{ auf } p \\ -c_f(p) & \text{falls } (v, u) \text{ auf } p \\ 0 & \text{sonst} \end{cases}$$

Augmentiere f entlang p: $f := f + f_p$

f_p ist ein Fluß in G_f

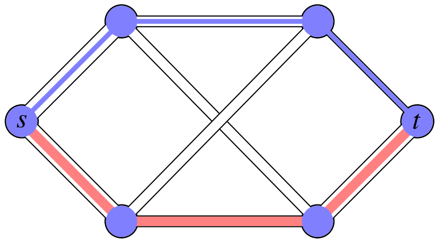
Die Ford–Fulkerson–Methode



Anfangs ist der Fluß 0.

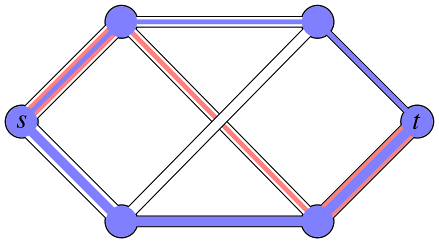
Der augmentierende Pfad ist rot eingezeichnet.

Die Ford–Fulkerson–Methode



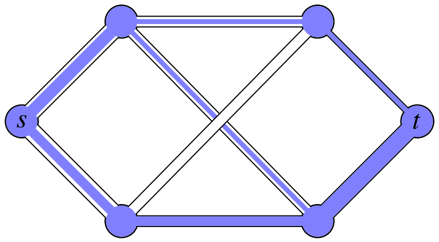
Der augmentierende Pfad hat Kapazität 5.

Die Ford–Fulkerson–Methode



Der augmentierende Pfad hat Kapazität 3.

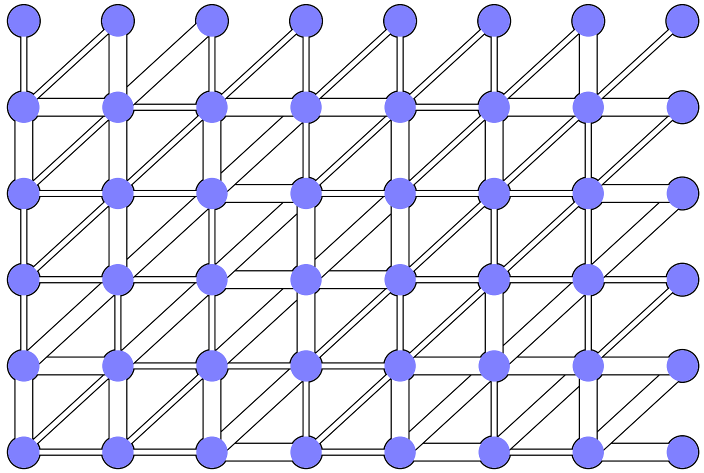
Die Ford–Fulkerson–Methode



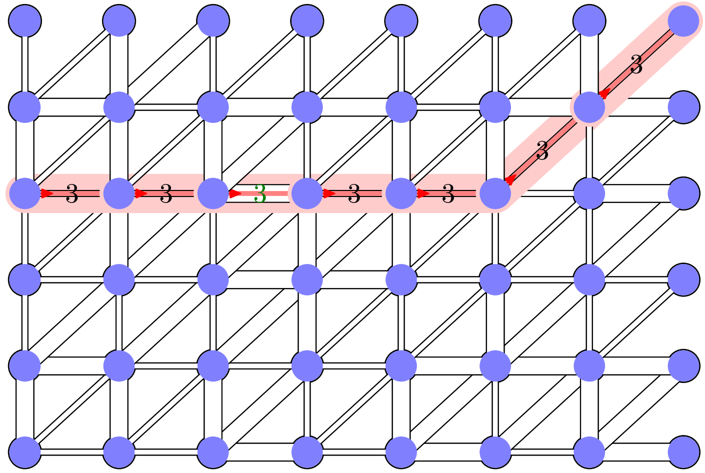
Jetzt gibt es keinen augmentierenden Pfad mehr.

Der Fluß ist maximal.

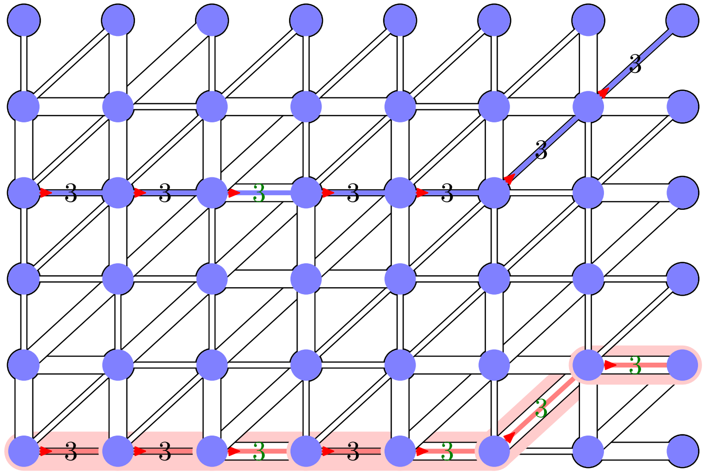
Beispiel



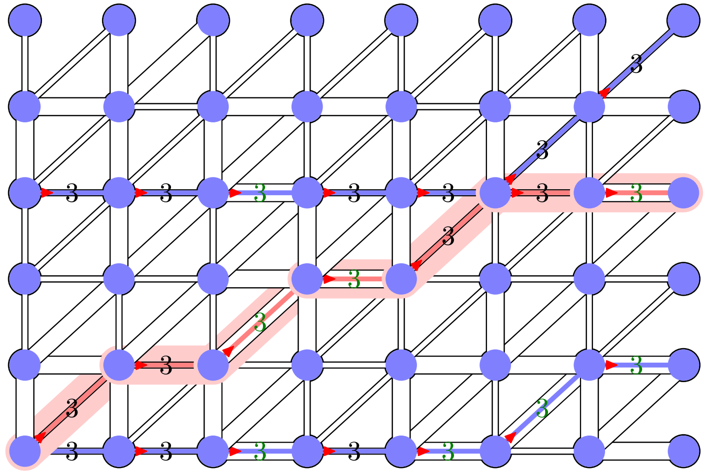
Beispiel



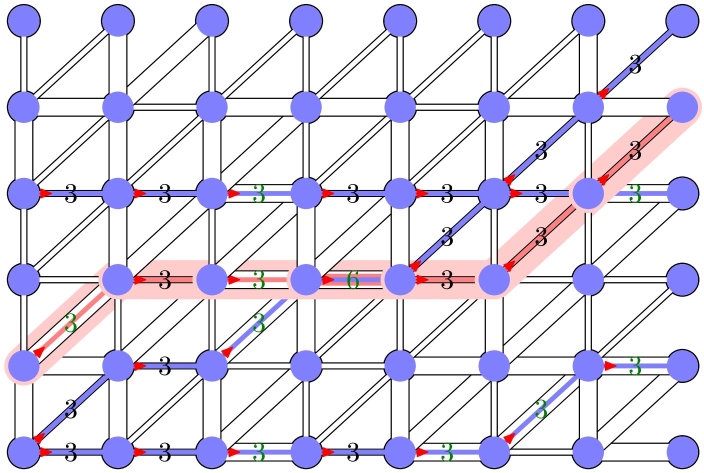
Beispiel



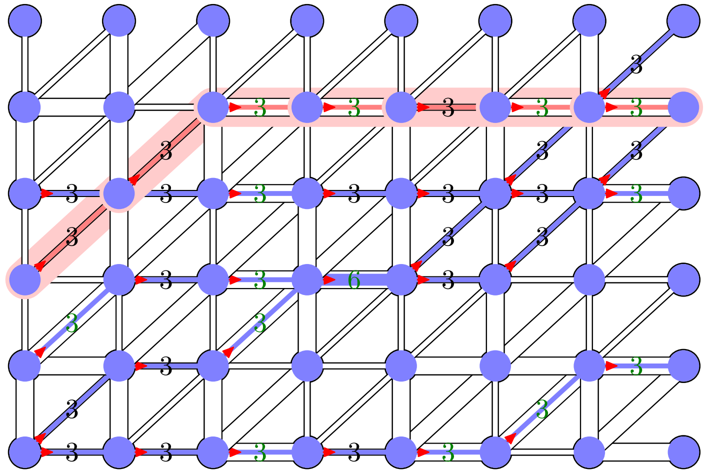
Beispiel



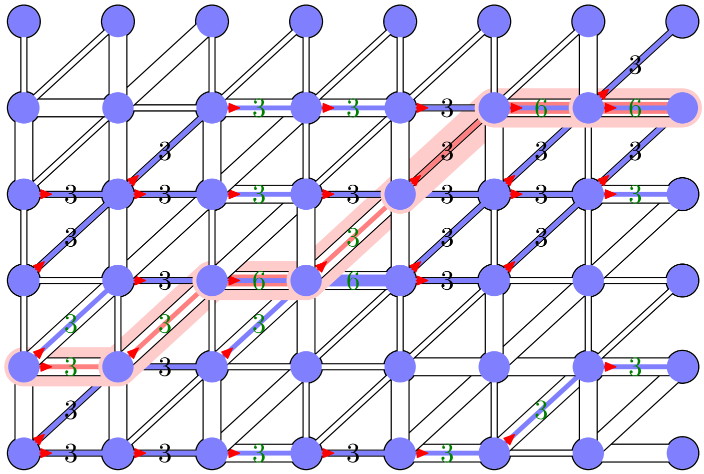
Beispiel



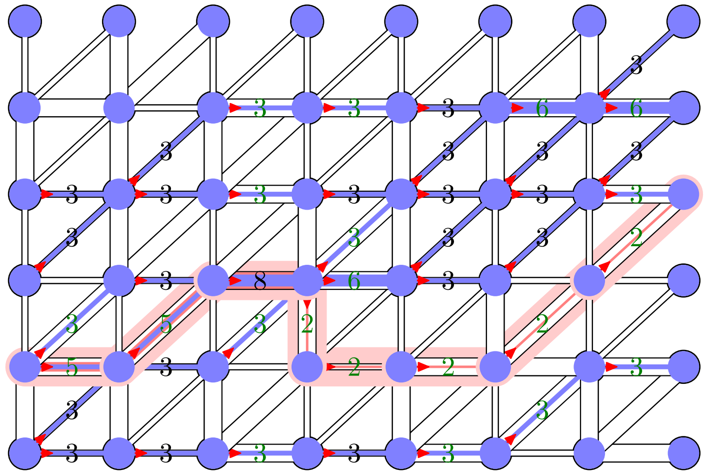
Beispiel



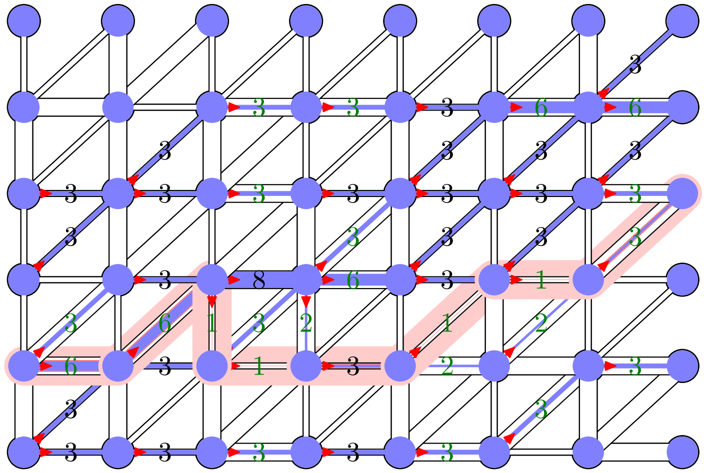
Beispiel



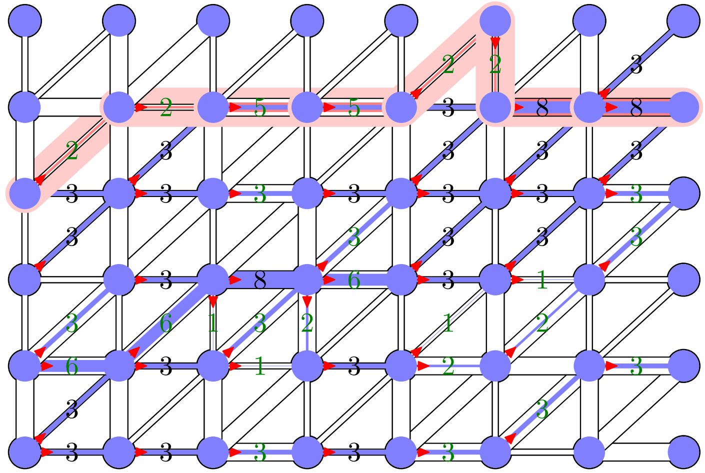
Beispiel



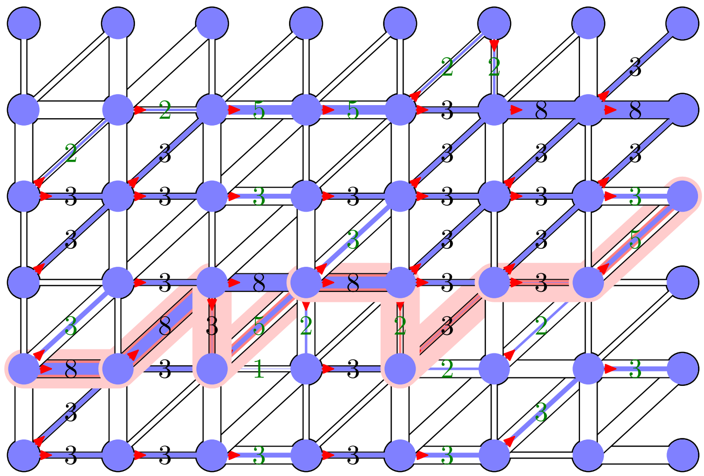
Beispiel



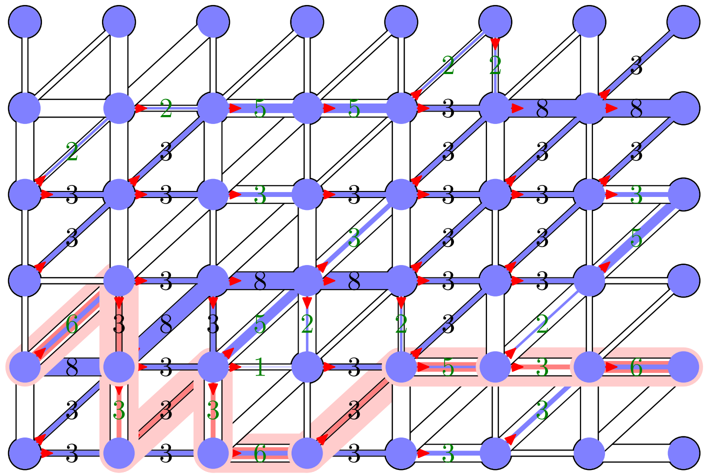
Beispiel



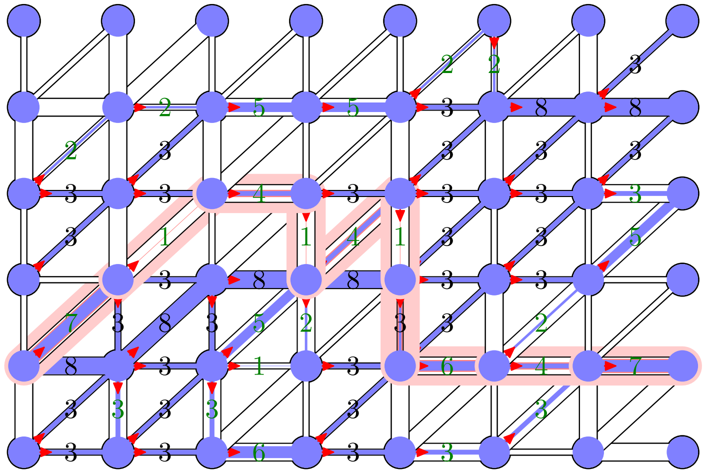
Beispiel



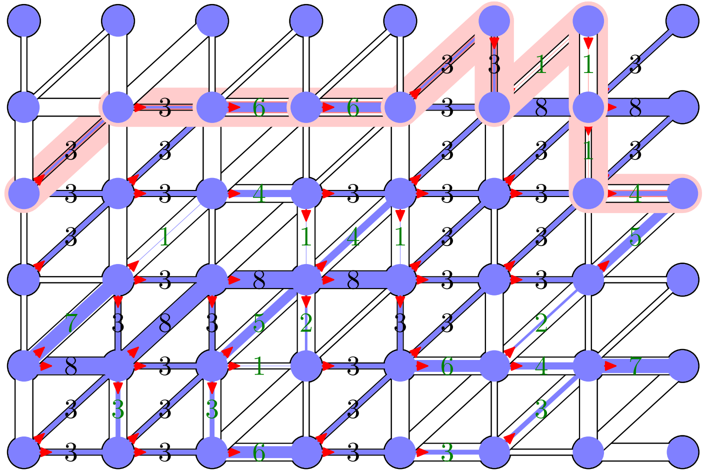
Beispiel



Beispiel



Beispiel



Korrektheit

Lemma B

Set $G = (V, E)$ ein s-t-Netzwerk und f ein Fluß in G.

Sei f' ein Fluß in G_f.

Dann ist $f + f'$ ein Fluß in G.

Konsequenz:

Die Ford–Fulkerson–Methode berechnet einen Fluß.

Beweis:

Wir müssen zeigen, daß $f + f'$ zulässig, symmetrisch und flußerhaltend ist.

Beweis (Symmetrie)

$$
\begin{aligned}
(f + f')(u, v) &= f(u, v) + f'(u, v) \\
&= -f(v, u) - f'(v, u) \\
&= -(f(v, u) + f'(v, u)) \\
&= -(f + f')(v, u)
\end{aligned}
$$

Beweis (Flußerhaltung)

Sei $u \in V - \{s,t\}$.

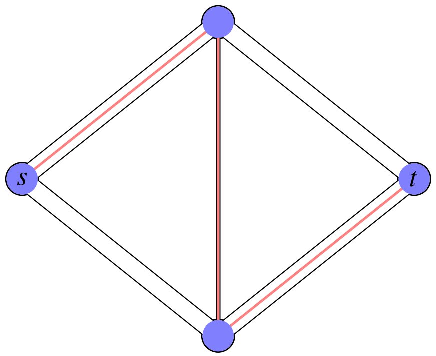
$$\begin{aligned}(f+f')(u,V) &= f(u,V)+f'(u,V)\\ &= 0+0\\ &= 0\end{aligned}$$

Beweis (Zulässigkeit)

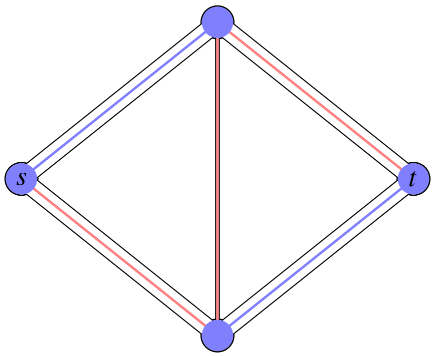
$$\begin{aligned}(f+f')(u,v) &= f(u,v)+f'(u,v)\\ &\le f(u,v)+c_f(u,v)\\ &= f(u,v)+(c(u,v)-f(u,v))\\ &= c(u,v)\end{aligned}$$

Der Beweis verwendet, daß f' ein Fluß in G_f ist, aber nicht, daß f ein Fluß in G ist.

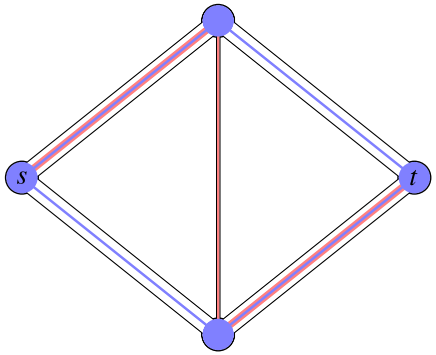
Laufzeit der Ford–Fulkerson–Methode



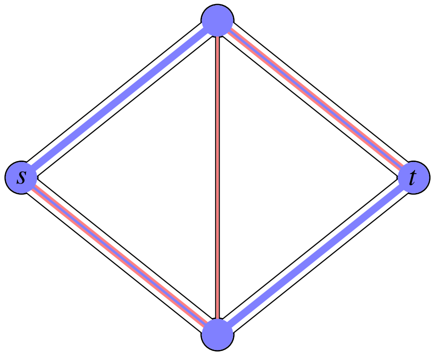
Laufzeit der Ford–Fulkerson–Methode



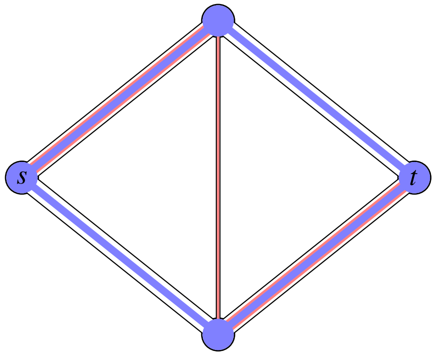
Laufzeit der Ford–Fulkerson–Methode



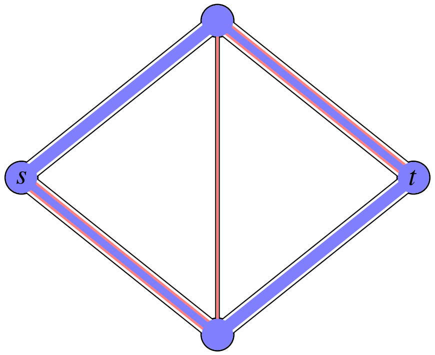
Laufzeit der Ford–Fulkerson–Methode



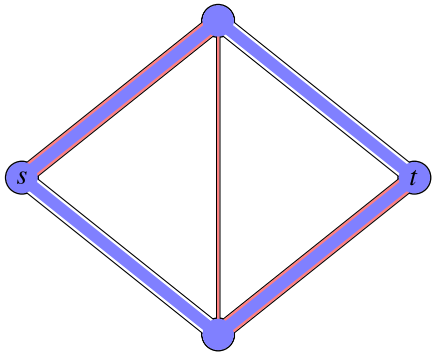
Laufzeit der Ford–Fulkerson–Methode



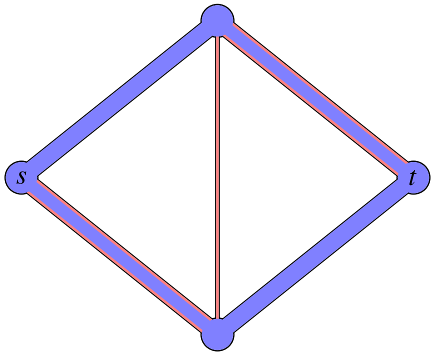
Laufzeit der Ford–Fulkerson–Methode



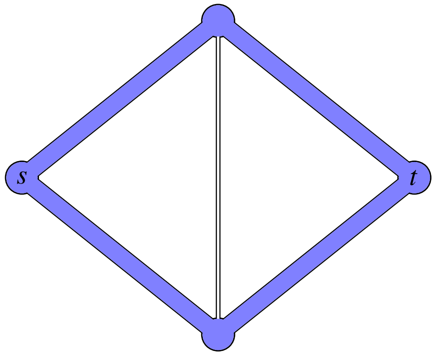
Laufzeit der Ford–Fulkerson–Methode



Laufzeit der Ford–Fulkerson–Methode



Laufzeit der Ford–Fulkerson–Methode



Laufzeit der Ford–Fulkerson–Methode

Ein Flußproblem ist *integral*, wenn alle Kapazitäten ganzzahlig sind.

Theorem.

Die Ford–Fulkerson–Methode benötigt nur $O(f^*)$ Iterationen, um ein integrales Flußproblem zu lösen, falls der Betrag eines maximalen Flusses f^* ist.

Beweis.

In jeder Iteration wird der Betrag des Flusses um $c_f(p) \geq 1$ erhöht. Er ist anfangs 0 und am Ende f^*.

Korollar.

Bei rationalen Kapazitäten terminiert die Ford–Fulkerson–Methode.

Schnitte in Netzwerken

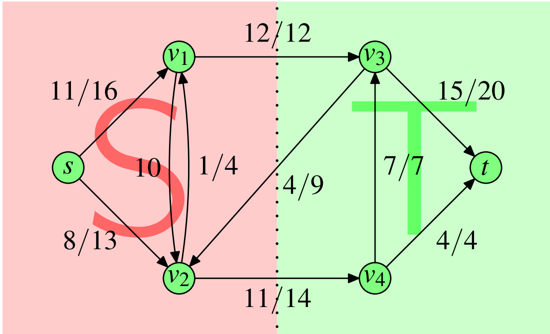
Ein *Schnitt* (S, T) in einem s-t-Netzwerk $G = (V, E)$ ist eine Partition $S \cup T = V$, $S \cap T = \emptyset$ mit $s \in S$ und $t \in T$.

Wenn f ein Fluß in G ist, dann ist $f(S, T)$ der *Fluß über* (S, T).

Die *Kapazität von* (S, T) ist $c(S, T)$.

Ein *minimaler Schnitt* ist ein Schnitt mit minimaler Kapazität.

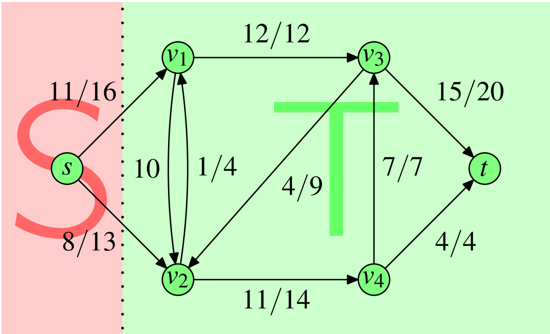
Schnitte in Netzwerken



Der Fluß über (S, T) ist 19.

Die Kapazität von (S, T) ist 26.

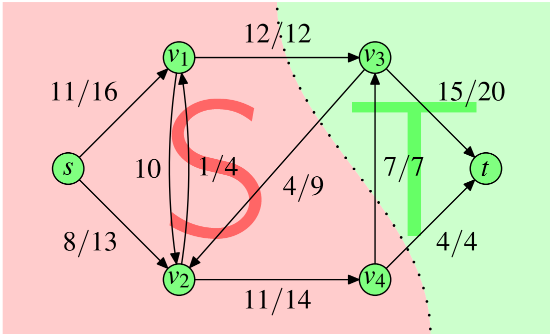
Schnitte in Netzwerken



Der Fluß über (S, T) ist 19.

Die Kapazität von (S, T) ist 29.

Schnitte in Netzwerken



Der Fluß über (S, T) ist 19.

Die Kapazität von (S, T) ist 23.

Fluß über einen Schnitt

Lemma C

Der Fluß über einen Schnitt und der Betrag des Flusses sind identisch, d.h. $f(S,T) = |f|$.

Beweis

$$\begin{aligned} f(S,T) &= f(S,V) - f(S,V-T) = f(S,V) - f(S,S) \\ &= f(S,V) = f(s,V) + f(S-s,V) \\ &= f(s,V) = |f| \end{aligned}$$

Spezialfälle

$|f| = f(s,V-s) = f(V-t,t)$

Max-flow Min-cut Theorem

Theorem

Sei f ein Fluß im s-t-Netzwerk $G = (V,E)$.

Dann sind äquivalent:

1. f ist ein maximaler Fluß
2. In G_f gibt es keinen augmentierenden Pfad
3. $|f| = c(S,T)$ für einen Schnitt (S,T)

Folgerungen

Falls die Ford–Fulkerson–Methode terminiert, berechnet sie einen maximalen Fluß.

Die Kapazität eines kleinsten Schnittes gleicht dem Betrag eines größten Flusses.

Beweis 1. → 2.

Sei f ein maximaler Fluß.

Nehmen wir an, es gebe einen augmentierenden Pfad p.

Dann ist $f + f_p$ ein Fluß in G mit $|f + f_p| > |f|$.

Das ist ein Widerspruch zur Maximalität von f.

Beweis 2. → 3.

G_f hat keinen s-t-Pfad.

$S := \{ v \in V \mid \text{ es gibt einen } s\text{-}v\text{-Pfad in } G_f \}$

$T := V - S$

Dann ist (S,T) ein Schnitt und es gilt $f(u,v) = c(u,v)$ für alle $u \in S$, $v \in T$.

Nach Lemma C gilt dann $f(S,T) = c(S,T) = |f|$.

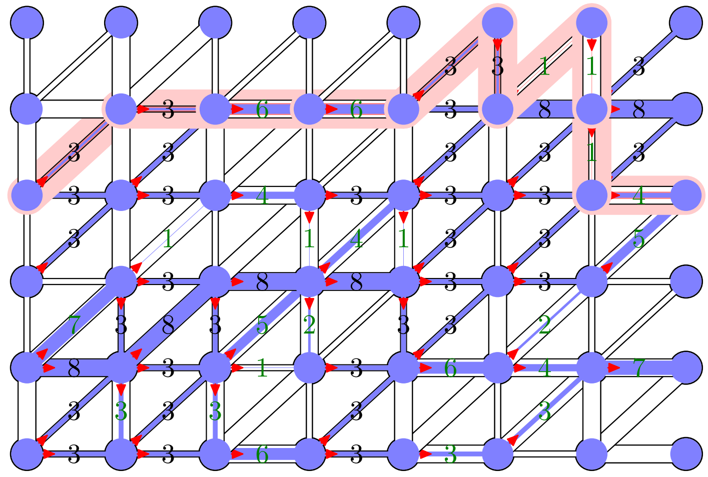
Beweis 3. → 1.

Sei f ein beliebiger Fluß.

$$\begin{aligned} |f| &= f(S,T) \\ &= \sum_{u \in S} \sum_{v \in T} f(u,v) \\ &\le \sum_{u \in S} \sum_{v \in T} c(u,v) \\ &= c(S,T) \end{aligned}$$

Der Betrag jedes Flusses ist also höchstens $c(S,T)$. Erreicht er sogar $c(S,T)$ ist er folglich maximal.

Wo ist ein minimaler Schnitt?



Wie findet man einen minimalen Schnitt?

1. Einen maximalen Fluß berechnen.
2. Eine Kante (u,v) ist *kritisch*, wenn $c(u,v) = f(u,v)$.
3. S besteht aus Knoten, die von s aus über unkritische Kanten erreicht werden können.
4. T besteht aus allen anderen Knoten.

Es gibt aber bessere, direkte Methoden!

Ganzzahlige Flüsse

Theorem

Wenn alle Kapazitäten ganzzahlig sind, dann findet die Ford–Fulkerson–Methode einen maximalen Fluß f, so daß alle $f(u,v)$ ganzzahlig sind.

Beweis

Induktion zeigt, daß die Kapazität eines augmentierenden Pfads ganzzahlig ist und $f(u,v)$ stets ganzzahlig bleiben.

Bipartites Matching

Gegeben:

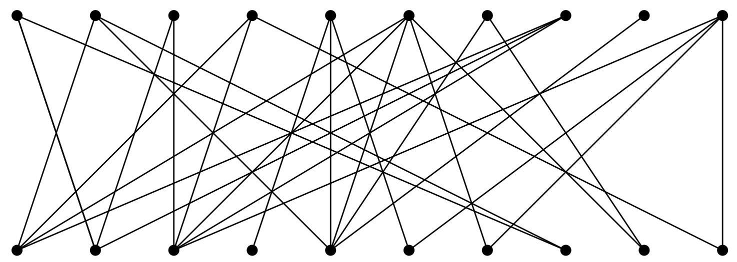
Ein bipartiter, ungerichteter Graph (V_1, V_2, E).

Gesucht:

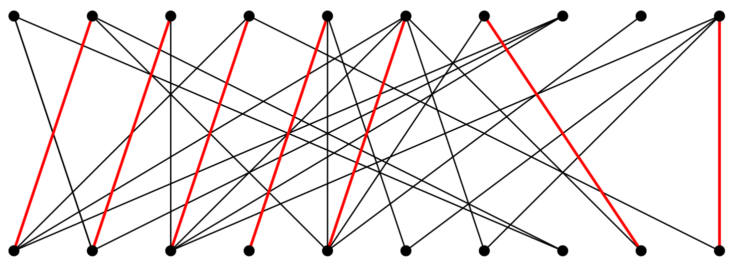
Ein Matching (Paarung) maximaler Kardinalität.

Ein *Matching* ist eine Menge paarweise nicht inzidenter Kanten, also $M \subseteq E$ mit $m_1, m_2 \in M,\ m_1 \neq m_2 \Rightarrow m_1 \cap m_2 = \emptyset$.

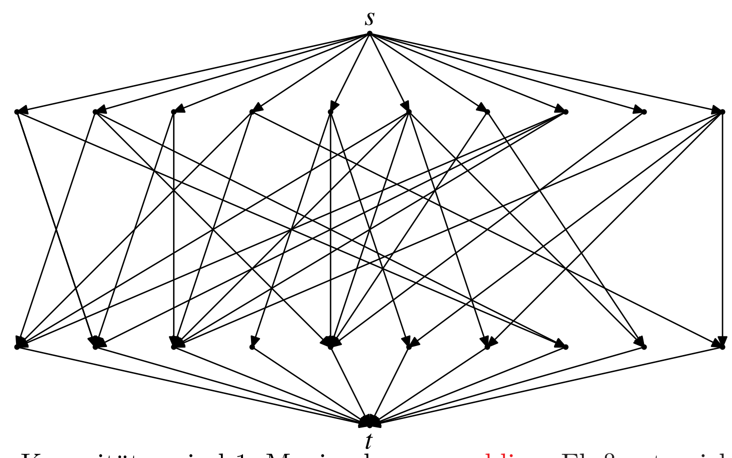
Beispiel



Beispiel

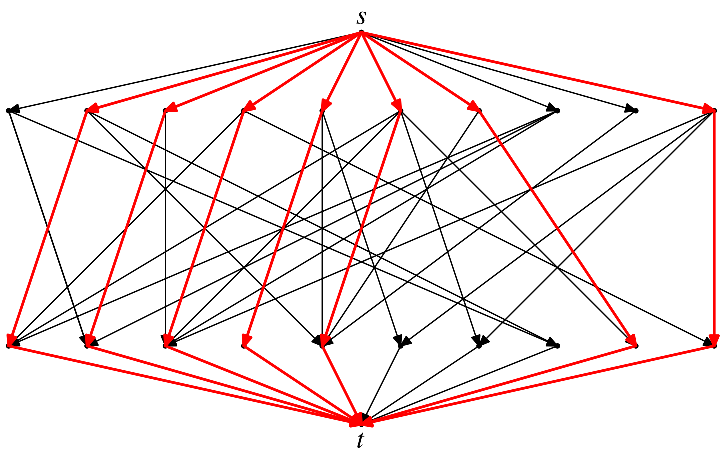


Lösung als Flußproblem

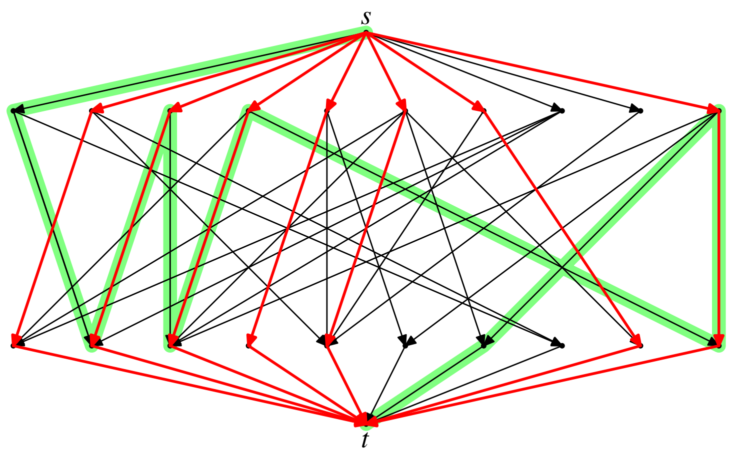


Alle Kapazitäten sind 1. Maximaler ganzzahliger Fluß entspricht einem Matching maximaler Kardinalität.

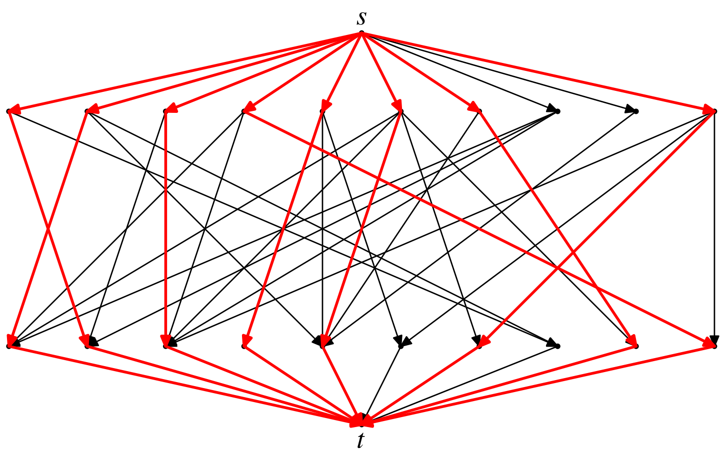
Gibt es einen augmentierenden Pfad?



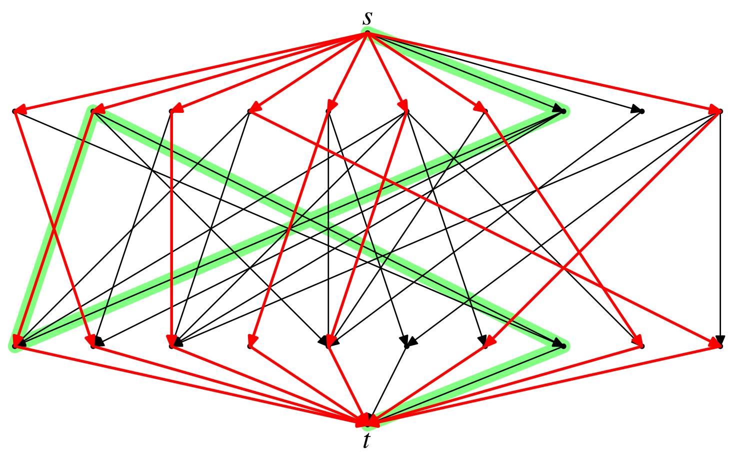
Gibt es einen augmentierenden Pfad? – Ja



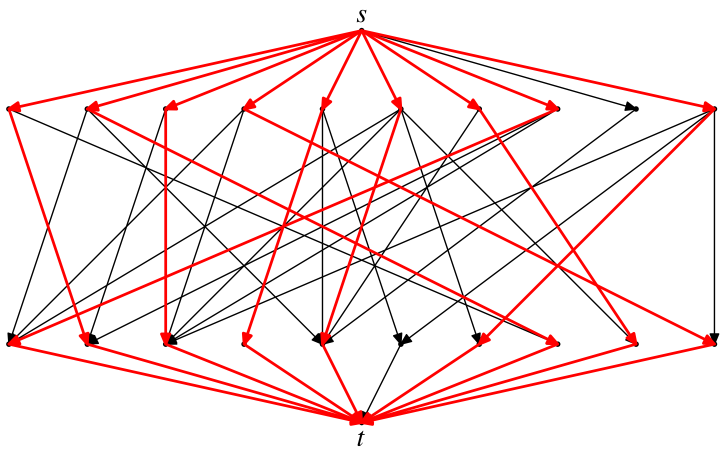
⇒ Neues Matching



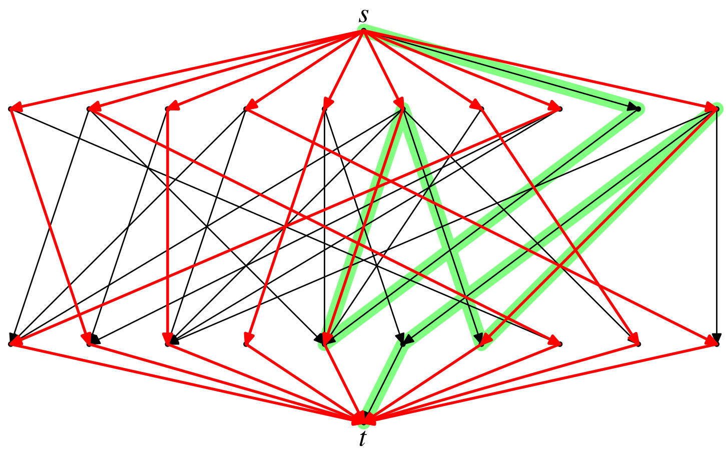
Es gibt wieder einen augmentierenden Pfad



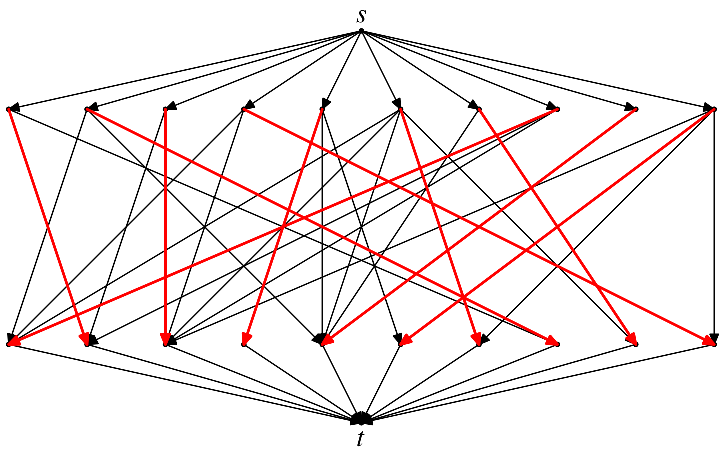
⇒ Neues Matching



Es gibt wieder einen augmentierenden Pfad



Ergebnis: Perfektes Matching



Laufzeit

Finden eines Matchings maximaler Kardinalität dauert nur $O(|E| \cdot \min\{|V_1|, |V_2|\})$ mit der Ford–Fulkerson–Methode.

- Der Fluß ist höchstens $f^* = \min\{|V_1|, |V_2|\}$.
- Finden eines Pfads dauert $O(|E|)$.

Variante der Ford–Fulkerson–Methode

Dieser Algorithmus funktioniert bei ganzzahligen Kapazitäten:

$K := 2^{\lfloor \log_2(\max\{ c(u,v) \mid (u,v) \in E \}) \rfloor}$
$f := 0$
while $K \geq 1$ **do**
  while es gibt einen augmentierenden Pfad p mit $c_f(p) \geq K$ **do**
    augmentiere f entlang p
  $K := K/2$
return f

Die Laufzeit ist $O(|E|^2 \log K)$.

$\Rightarrow$ vergleiche mit $O(|E| f^*)$.

Variante der Ford–Fulkerson–Methode

Theorem

Die Laufzeit dieser Variante beträgt $O(|E|^2 \log C)$, wobei $C = \max\{ c(u,v) \mid (u,v) \in E \}$.

Beweis

- Die Restkapazität eines minimalen Schnitts ist stets höchstens $2K|E|$.
- Für jedes K gibt es nur $|E|$ Augmentierungen
- Es gibt $O(\log C)$ verschiedene K

Der Edmonds–Karp–Algorithmus

Die Ford–Fulkerson–Methode kann sehr langsam sein, auch wenn das Netzwerk klein ist.

Der Edmonds–Karp–Algorithmus ist polynomiell in der Größe des Netzwerks.

Initialisiere Fluß f zu 0
while es gibt einen augmentierenden Pfad **do**
  finde einen kürzesten augmentierenden Pfad p
  augmentiere f entlang p
return f

Unterschied: Es wird ein kürzester Pfad gewählt

Der Edmonds–Karp–Algorithmus

for each edge $(u,v) \in E$ **do**
  $f(u,v) := 0$
  $f(v,u) := 0$
while there exists a path from s to t in G_f **do**
  $p :=$ a shortest path from s to t in G_f
  $c_f(p) := \min\{ c_f(u,v) \mid (u,v)$ is in $p \}$
  for each edge (u,v) in p **do**
    $f(u,v) := f(u,v) + c_f(p)$
    $f(v,u) := -f(u,v)$
return f

Analyse des Edmonds–Karp–Algorithmus

Lemma D

Gegeben ist ein s-t-Netzwerk $G = (V, E)$. Sei f ein Fluß, der während der Ausführung des Edmonds–Karp–Algorithmus vorkommt und f' der Fluß nach der anschließenden Augmentierung.

Dann gilt: $\delta'(s, v) \geq \delta(s, v)$ für alle $v \in V$.

Hierbei ist $\delta(u, v)$ (bzw. $\delta'(u, v)$) die Länge des kürzesten Pfads von u nach v in G_f (bzw. in $G_{f'}$).

Analyse des Edmonds–Karp–Algorithmus

Beweis

Durch Widerspruch. Nehmen wir an, es gibt ein v mit

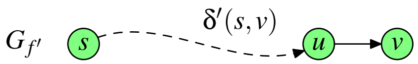
$$\delta(s, v) > \delta'(s, v). \tag{W}$$

Ohne Einschränkung können wir annehmen, daß

$$\delta(s, u) > \delta'(s, u) \Longrightarrow \delta'(s, u) \geq \delta'(s, v). \tag{M}$$

Wir müssen nur unter den möglichen v ein solches mit minimalem $\delta'(s, v)$ wählen.

Analyse des Edmonds–Karp–Algorithmus



u ist vorletzter Knoten auf kürzestem s-v-Pfad.

1. Möglichkeit: $(u, v) \in E[G_f]$

Dann gilt:

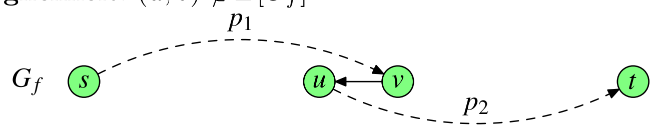
$$\delta'(s, v) = \delta'(s, u) + 1 \geq \delta(s, u) + 1 \text{ wegen (M)}$$

und

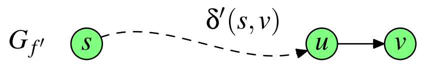
$$\delta(s, v) \leq \delta(s, u) + 1 \text{ wegen } (u, v) \in E[G_f]$$

Daraus folgt $\delta'(s, v) \geq \delta(s, v)$. Widerspruch zu (W).

2. Möglichkeit: $(u, v) \notin E[G_f]$



$p_1, (v, u), p_2$ wird als augmentierender Pfad benutzt.



Es gilt:

$$\delta(s, u) = \delta(s, v) + 1 \text{ und } \delta'(s, u) = \delta'(s, v) - 1$$

Wegen (M) gilt ausserdem:

$$\delta'(s, u) \geq \delta(s, u)$$

Zusammen folgt $\delta'(s, v) \geq \delta(s, v) + 2$, ein Widerspruch zu (W). □

Analyse des Edmonds–Karp–Algorithmus

Theorem

Der Edmonds–Karp–Algorithmus findet einen maximalen Fluß in $O(|E|^2|V|)$ Schritten.

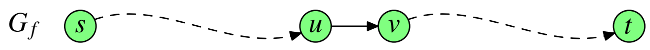
Beweis

- Als Spezialfall der Ford–Fulkerson–Methode ist der Algorithmus korrekt.
- Jede Augmentierung benötigt $O(|E|)$ Schritte. Ein kürzester augmentierender Pfad kann mit BFS gefunden werden.
- Wir werden zeigen, daß nur $O(|E| \cdot |V|)$ Augmentierungen vorgenommen werden.

Sei f der Fluß vor und f' der Fluß nach einer Augmentierung.

Eine Kante (u, v) heißt *kritisch*, wenn $f(u, v) < c(u, v)$ und $f'(u, v) = c(u, v)$.

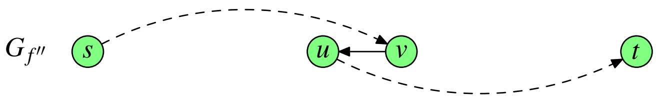
Jeder augmentierende Pfad enthält eine kritische Kante:



Sei (u, v) kritisch. Kann (u, v) später nocheinmal kritisch werden?

(u, v) verschwindet aus dem Residualnetzwerk.

Bevor (u, v) wieder kritisch wird, muß (u, v) wieder im Residualnetzwerk erscheinen:



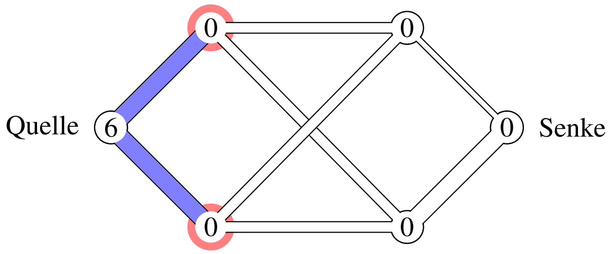
$\Rightarrow$ Der augmentierende Pfad geht durch (v, u).

- $\delta''(s, u) = \delta''(s, v) + 1$
- $\delta(s, v) = \delta(s, u) + 1$
- $\delta''(s, v) \geq \delta(s, v)$ wegen Lemma D
- $\Rightarrow \delta''(s, u) > \delta(s, u)$.

Bevor (u, v) wieder kritisch werden kann, hat sich der Abstand von s zu u erhöht.

Also kann eine Kante nur $|V|$ mal kritisch werden und es gibt höchstens $|E| \cdot |V|$ Augmentierungen. □

Preflow-Push-Algorithmen

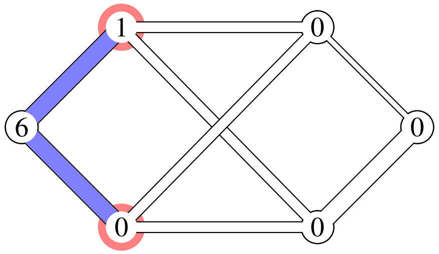


Überfließende Knoten sind rot markiert.

Die Zahlen geben die *Höhe* an.

(Die Kapazitäten sind wieder 3, 5 und 8.)

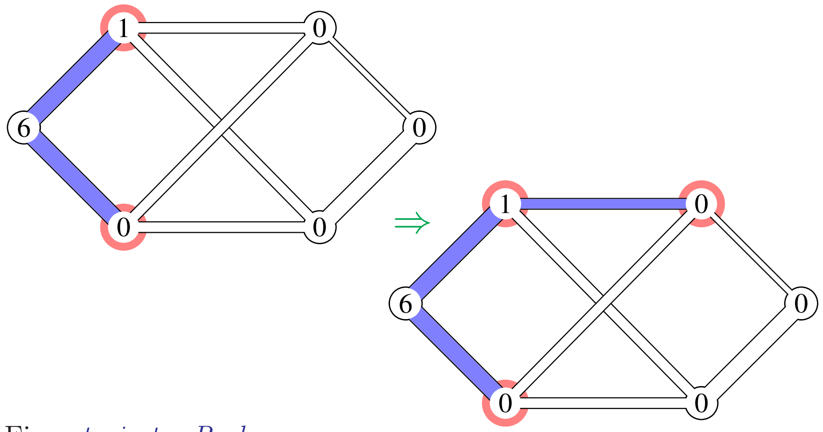
Preflow-Push-Algorithmen



Ein *Lift-Operation*.

Falls alle Nachbarn eines überfließender Knoten im Residualnetzwerk nicht kleinere Höhe haben, darf der Knoten „geliftet“ werden. Seine neue Höhe ist um eins höher als die minimale Höhe seiner Nachbarn im Residualnetzwerk.

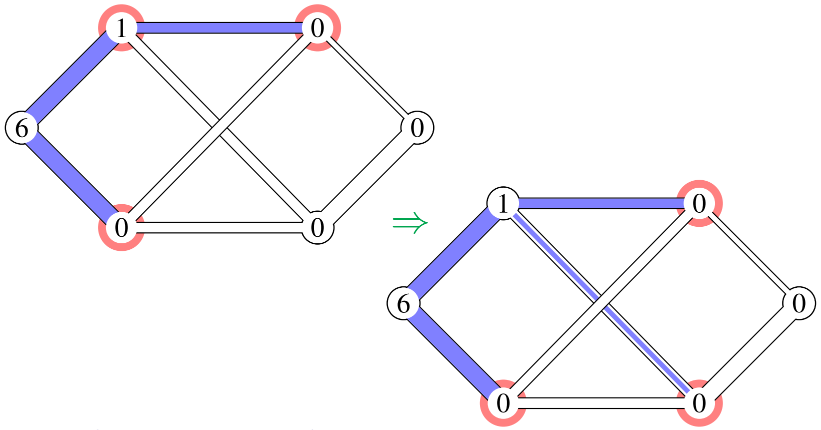
Preflow-Push-Algorithmen



Ein *saturierter Push*.

Zu einem Nachbarn im Residualnetzwerk kann Fluß „gepusht“ werden. Die Höhe muß dorthin um 1 abnehmen.

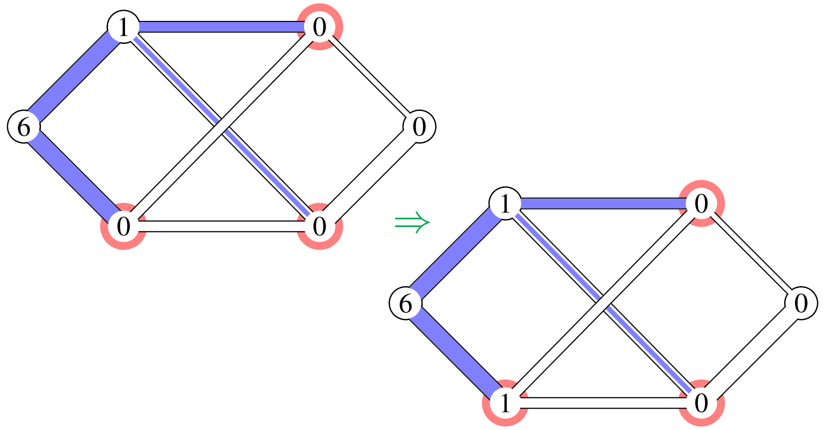
Preflow-Push-Algorithmen



Ein *nicht-saturierter Push*.

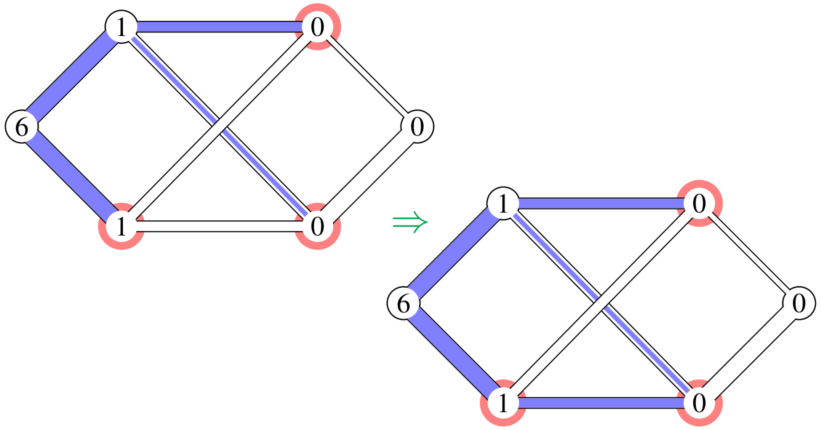
Ein „Push“ ist *saturiert*, wenn die Restkapazität der Kante erschöpft wird.

Preflow-Push-Algorithmen



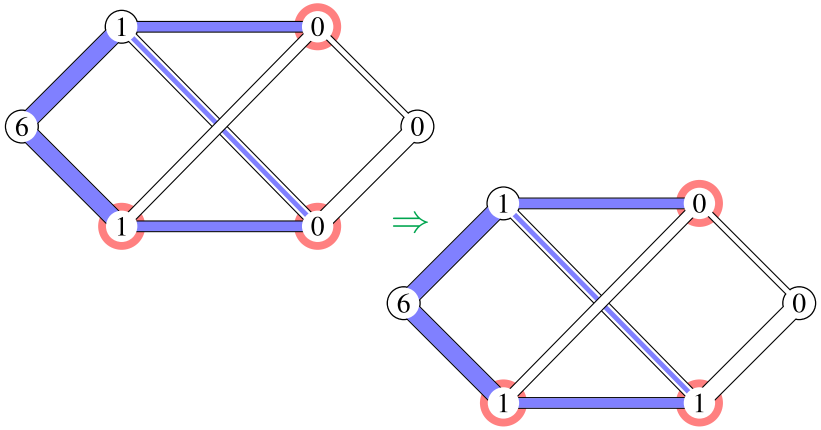
Welche Operationen sind hier auf die überfließenden Knoten anwendbar?

Preflow-Push-Algorithmen



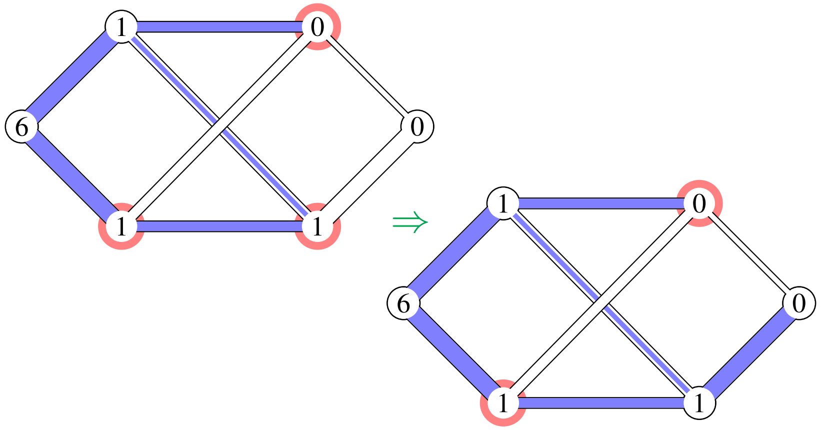
Es wurde ein saturierter Push durchgeführt.

Preflow-Push-Algorithmen



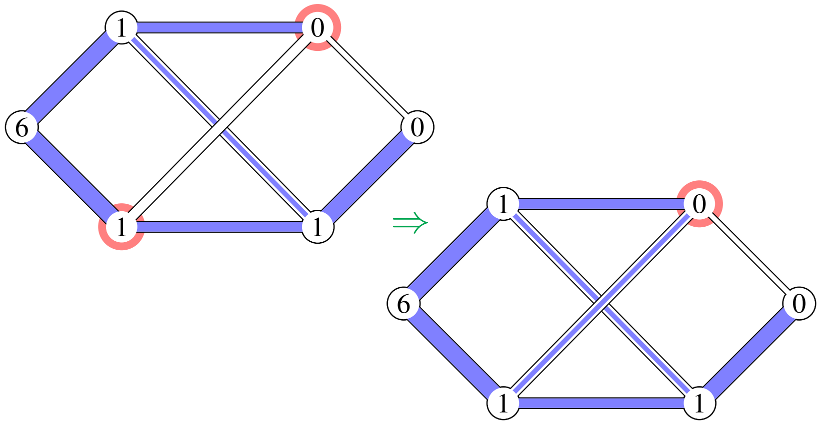
Es wurde eine Lift-Operation durchgeführt.

Preflow-Push-Algorithmen



Es wurde ein saturierter Push durchgeführt.

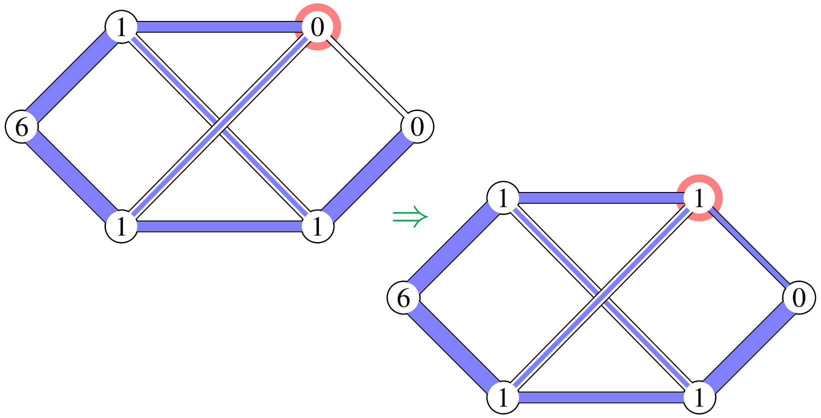
Preflow-Push-Algorithmen



Es wurde ein nicht-saturierter Push durchgeführt.

Jetzt gibt es nur eine Wahlmöglichkeit.

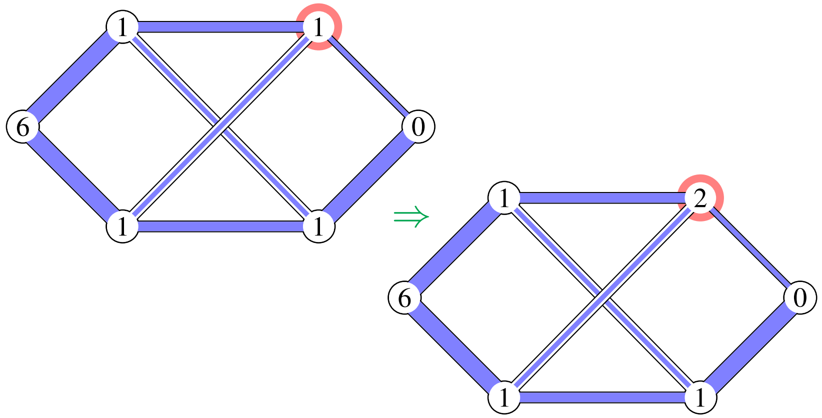
Preflow-Push-Algorithmen



Der Knoten wurde auf Höhe 1 geliftet.

Anschließend wurde ein Push durchgeführt.

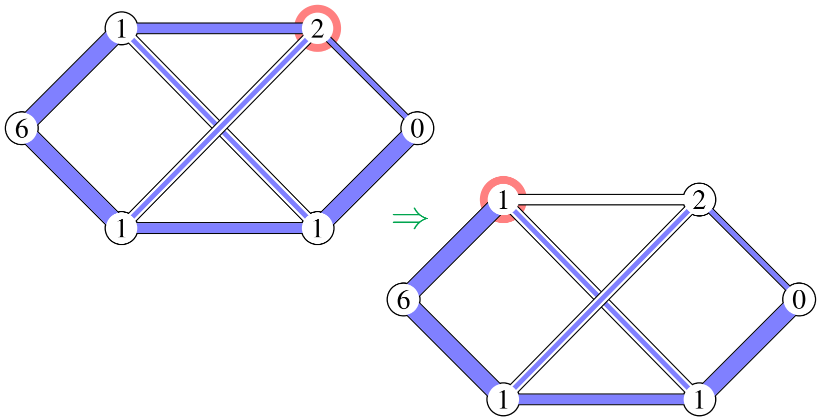
Preflow-Push-Algorithmen



Wir mußten den Knoten auf Höhe 2 liften.

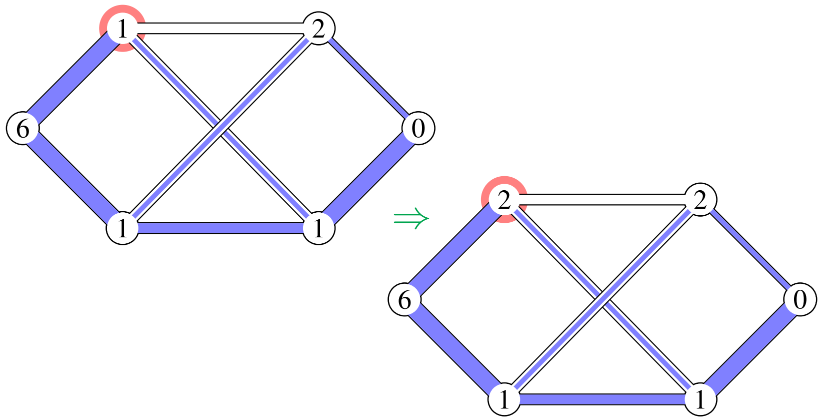
Es gibt jetzt zwei anwendbare Push-Operationen.

Preflow-Push-Algorithmen



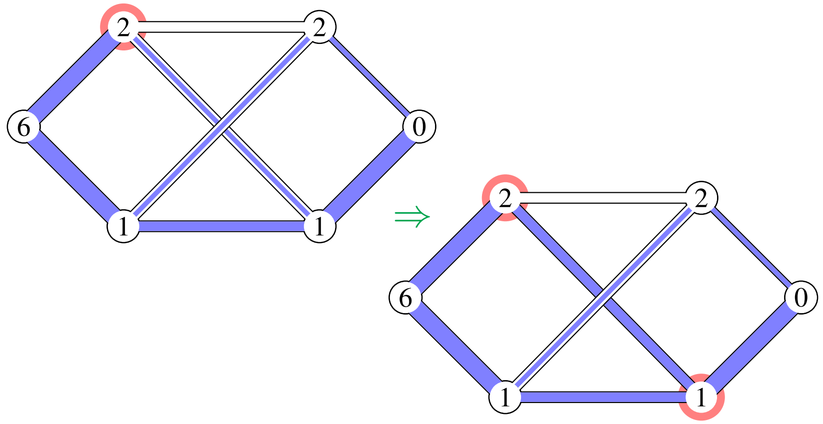
Durch einen saturierenden Push, wurde Fluß ausgelöscht.

Preflow-Push-Algorithmen



Eine Lift-Operation.

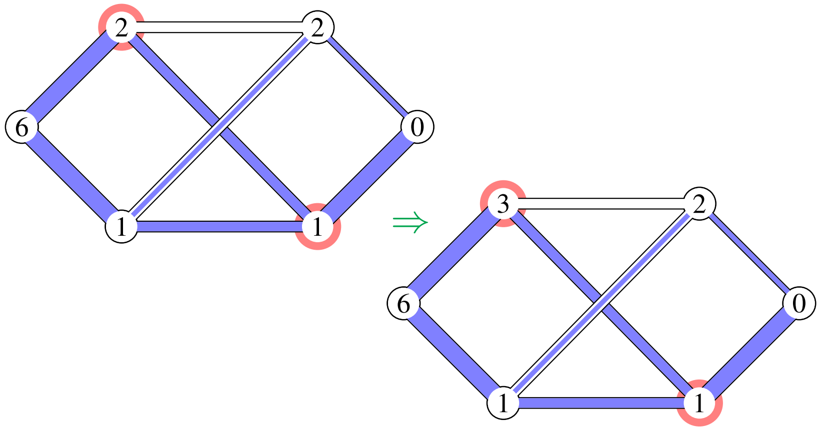
Preflow-Push-Algorithmen



Es wurde ein saturierter Push durchgeführt.

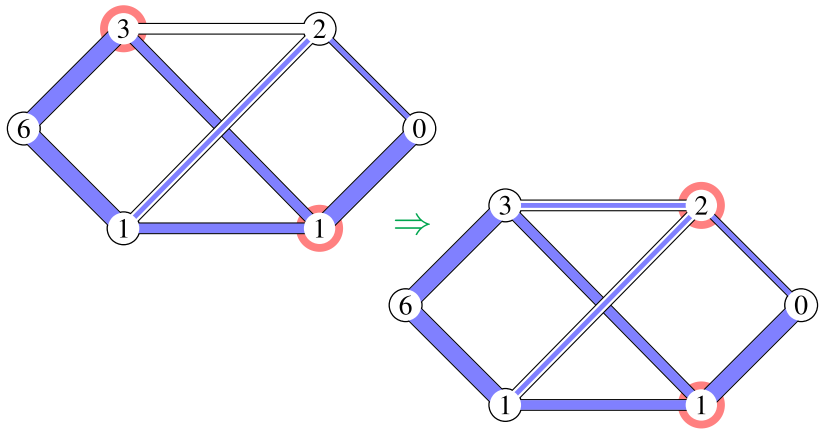
Dadurch gibt es jetzt zwei überfließende Knoten.

Preflow-Push-Algorithmen



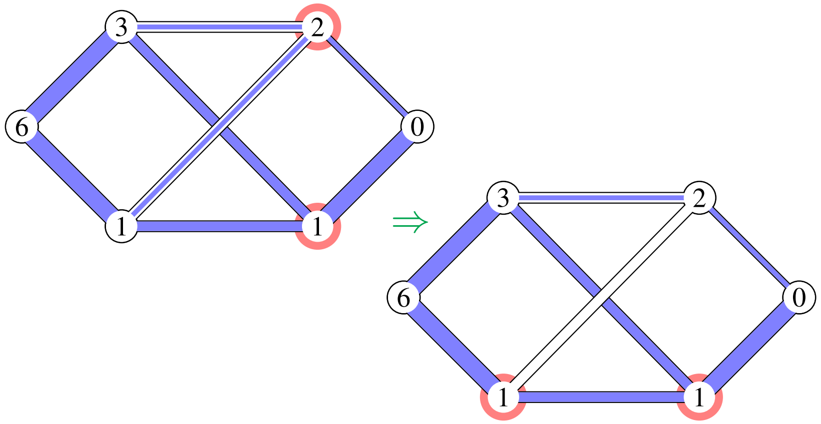
Ein Knoten wurde auf Höhe 3 geliftet.

Preflow-Push-Algorithmen



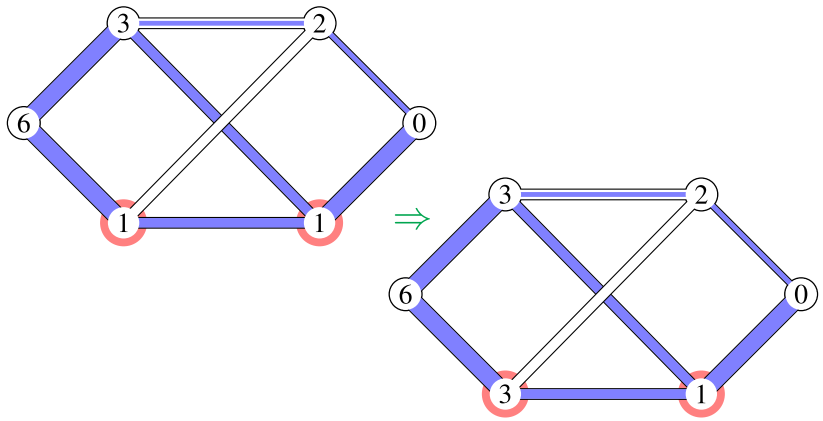
Es wurde ein nicht-saturierter Push durchgeführt.

Preflow-Push-Algorithmen

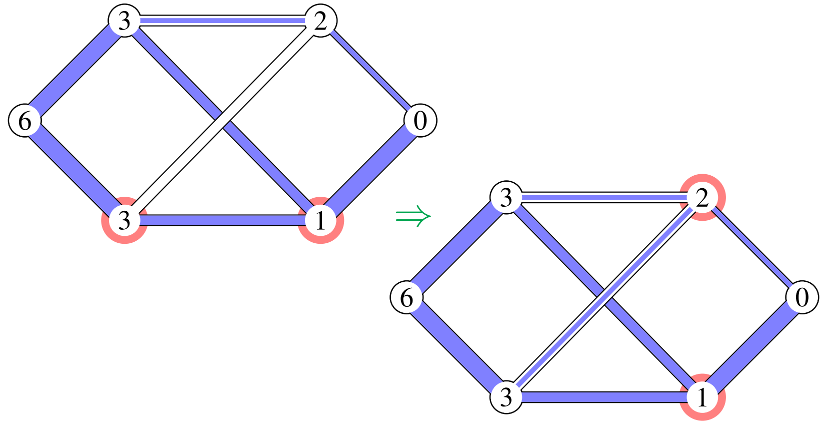


Es wurde ein saturierter Push durchgeführt.

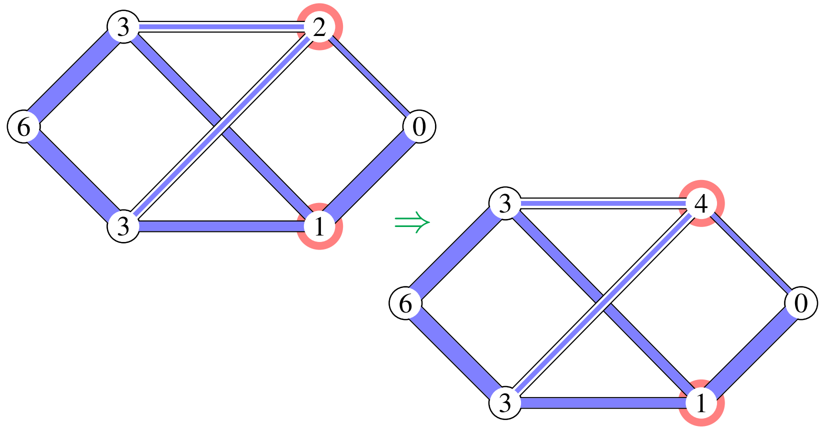
Preflow-Push-Algorithmen



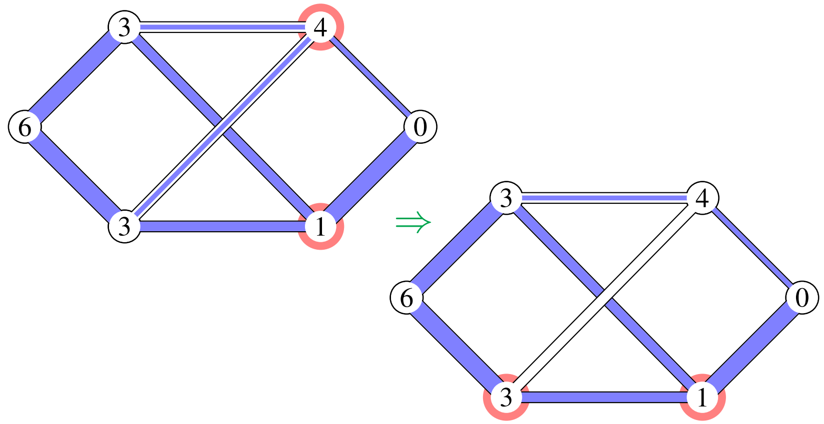
Preflow-Push-Algorithmen



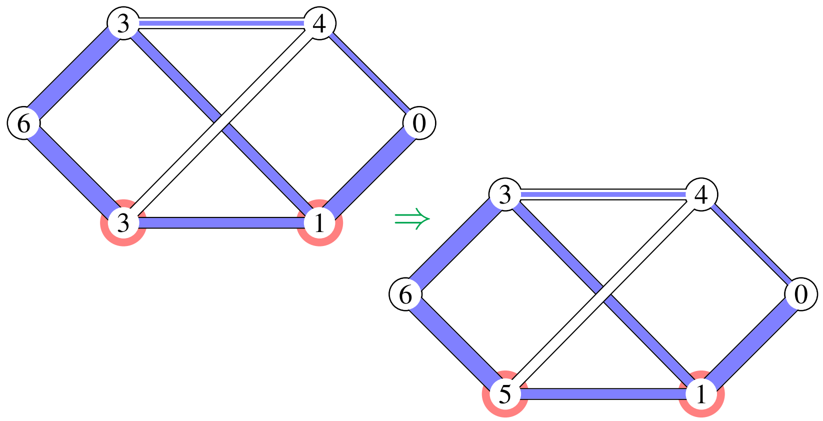
Preflow-Push-Algorithmen



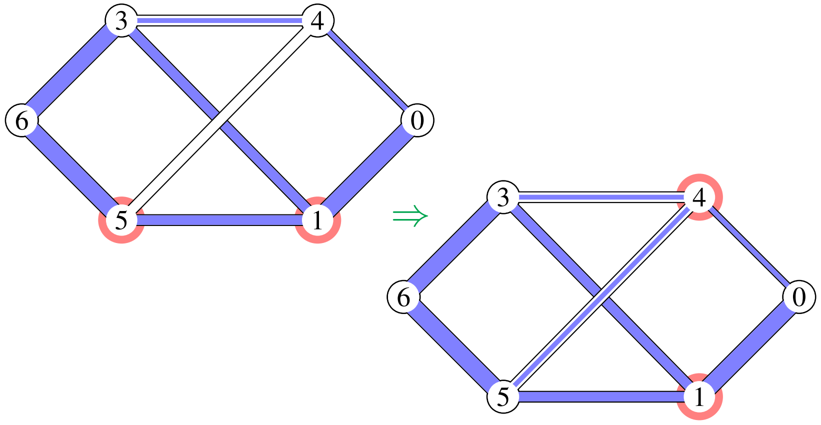
Preflow-Push-Algorithmen



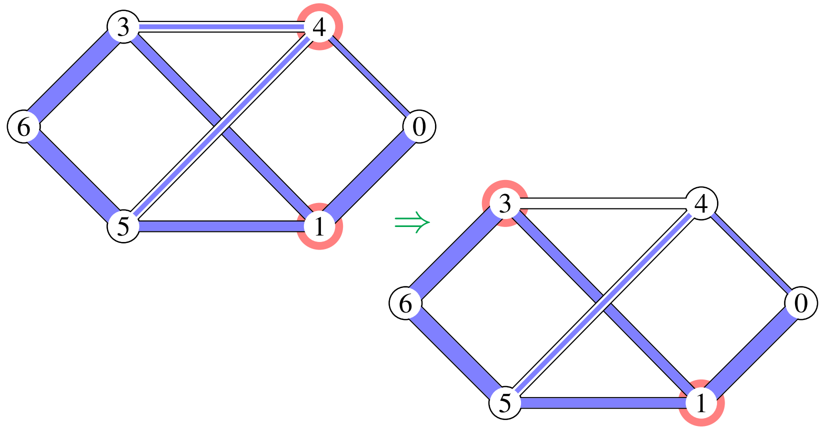
Preflow-Push-Algorithmen



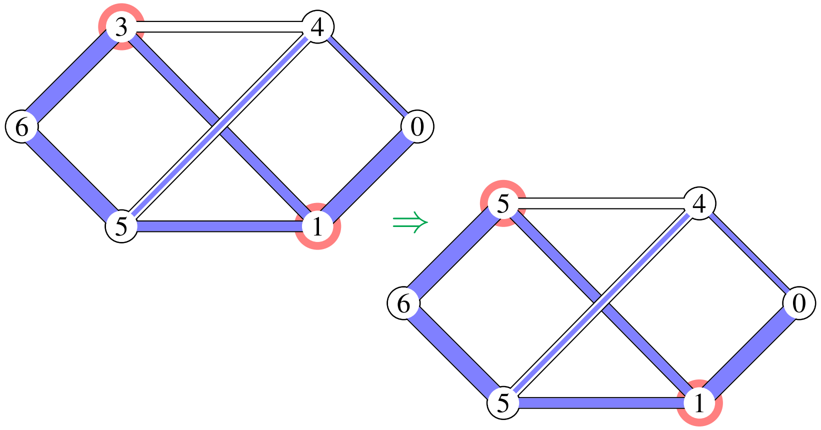
Preflow-Push-Algorithmen



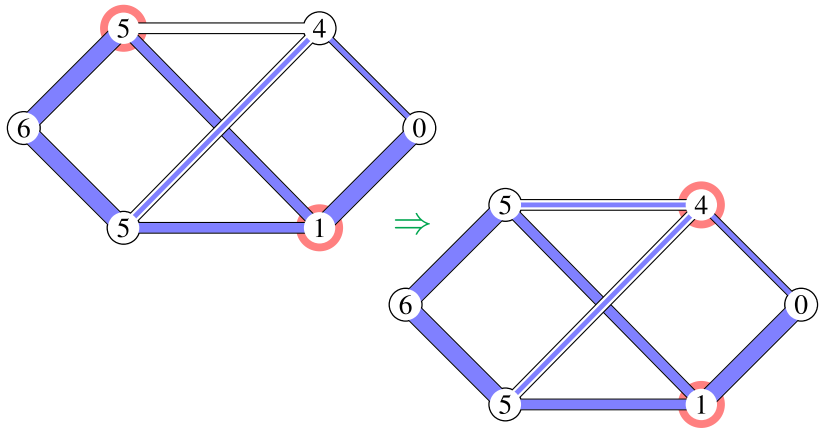
Preflow-Push-Algorithmen



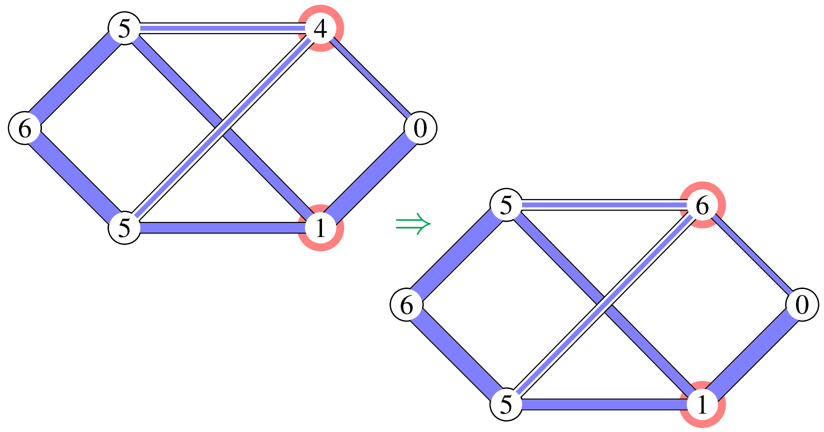
Preflow-Push-Algorithmen



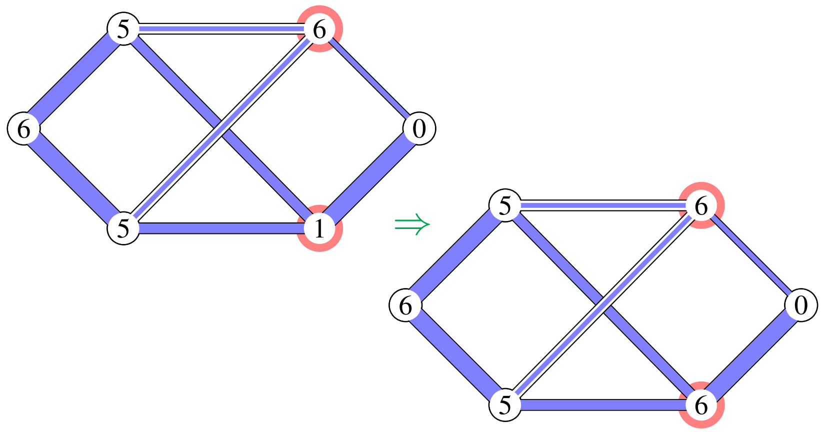
Preflow-Push-Algorithmen



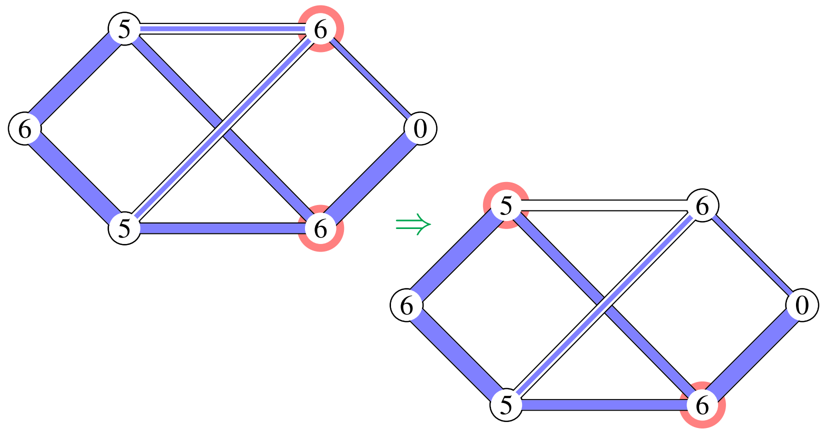
Preflow-Push-Algorithmen



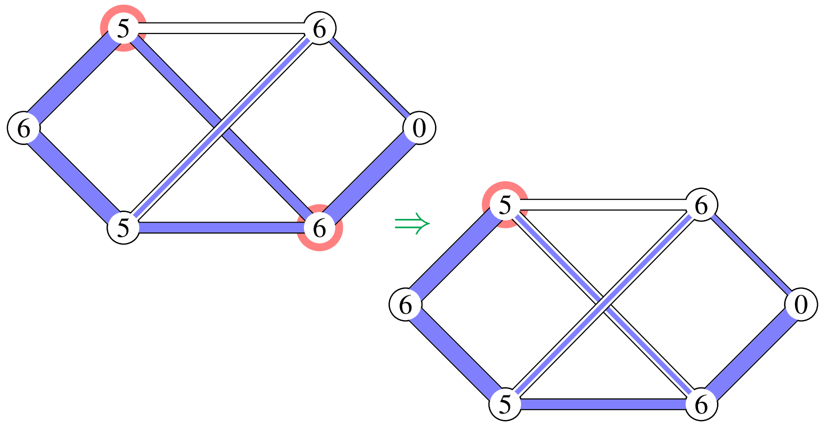
Preflow-Push-Algorithmen



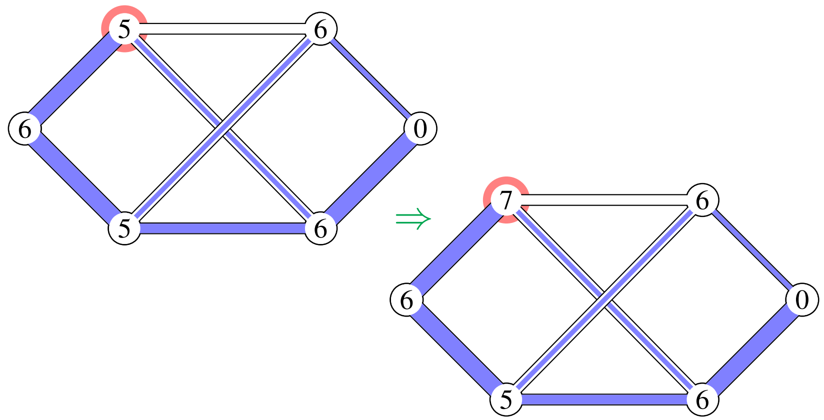
Preflow-Push-Algorithmen



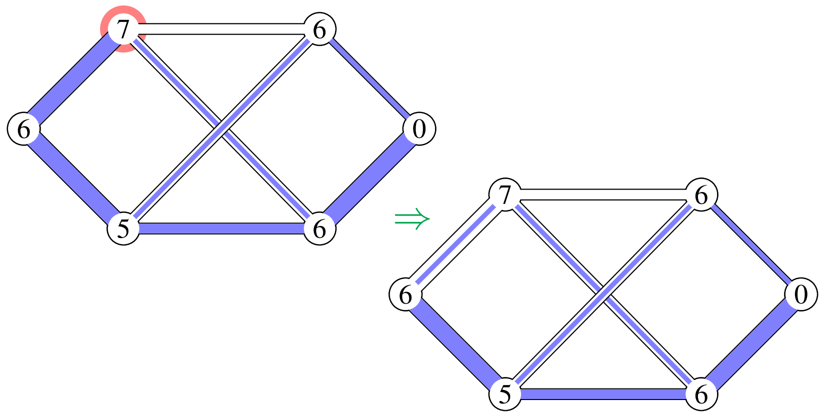
Preflow-Push-Algorithmen



Preflow-Push-Algorithmen



Preflow-Push-Algorithmen



- Es gibt keinen überfließenden Knoten mehr.
- Der „preflow“ ist jetzt ein echter (maximaler) Fluß.

Preflows

Ein *preflow* ist eine Funktion $f\colon V \times V \to \mathbf{R}$, die Paare von Knoten auf reelle Zahlen abbildet und diese Bedingungen erfüllt:

- *Zulässigkeit:* Für $u, v \in V$ gilt $f(u, v) \leq c(u, v)$.
- *Symmetrie:* Für $u, v \in V$ gilt $f(u, v) = -f(v, u)$.
- *statt Flußerhaltung:* Für $u \in V - \{s, t\}$ gilt $f(V, u) \geq 0$.

Wir definieren $e(u) = f(V, u)$, den *Exzeß* von u.

Falls $e(u) > 0$ dann ist u *überfließend*.

Höhenfunktionen

Definition

Sei $G = (V, E)$ ein s-t-Netzwerk und f ein Preflow auf G.

Eine Funktion $h\colon V \to \mathbf{N}$ ist eine *Höhenfunktion*, wenn

- $h(s) = |V|$
- $h(t) = 0$
- $h(u) \leq h(v) + 1$ für alle $(u, v) \in E_f$

„Eine Kante im Residualnetzwerk führt höchstens um 1 nach unten.“

Die Push-Operation $PUSH(u, v)$

Vorbedingungen:

- $e(u) > 0$
- $c_f(u, v) > 0$
- $h(u) = h(v) + 1$

$d_f(u,v) := \min\{e(u), c_f(u,v)\}$
$f(u,v) := f(u,v) + d_f(u,v)$
$f(v,u) := -f(u,v)$
$e(u) := e(u) - d_f(u,v)$
$e(v) := e(v) + d_f(u,v)$

Die Lift-Operation $LIFT(u)$

Vorbedingungen:

- $e(u) > 0$
- $h(u) \leq h(v)$ falls $(u, v) \in E_f$

$h(u) := 1 + \min\{\, h(v) \mid (u, v) \in E_f \,\}$

Der einfache Preflow-Push-Algorithmus

Eingabe: Ein s-t-Netzwerk $G = (V, E)$

for each $u \in V$ **do** $h(u) := 0,\ e(u) := 0$
$h(s) := |V|$
for each $(u, v) \in E$ **do** $f(u, v) := 0,\ f(v, u) := 0$
for each $u \in \{\, u \mid (s, u) \in E \}$ **do**
    $f(s, u) := c(s, u)$
    $f(u, s) := -c(s, u)$
    $e(u) := c(s, u)$
while es gibt eine anwendbare Lift-
        oder Push-Operation **do**
    wähle eine Operation aus und wende sie an
return f

Korrektheit

Lemma E

Während der Ausführung des Preflow-Push-Algorithmus ist h eine Höhenfunktion.

Beweis

Induktion über Anzahl der Operationen.

Anfangs führen keine Kanten nach unten ausser von s. Diese sind aber keine Residualkanten.

Beweis (Fortsetzung)

1. *LIFT*(u): Es gibt kein Problem mit einer Kante, wenn ihr Endpunkt gehoben wird.

   Sei also $(u, v) \in E_f$. Vorher galt $h(u) \leq h(v)$. Danach muß $h(u) \leq h(v) + 1$ gelten.

2. *PUSH*(u, v): Es kann eine neue Kante $(v, u) \in E_f$ entstehen. Vorher galt aber $h(u) = h(v) + 1$, also führt die neue Kante sogar um eins nach oben.

Korrektheit

Lemma F

Sei $G = (V, E)$ ein s-t-Netzwerk, f ein Preflow und h eine Höhenfunktion auf G.

Dann gibt es keinen s-t-Pfad in G_f.

Beweis

Gäbe es einen s-t-Pfad, hätte er höchstens Länge $|V| - 1$. Jede Kante führt um höchstens eins nach unten. Es startet auf Höhe $|V|$.

Also kann er nicht auf Höhe 0 enden. Widerspruch.

Korrektheit

Theorem

Der Preflow-Push-Algorithmus berechnet einen maximalen Fluß, falls er terminiert.

Beweis

Wenn er terminiert, gibt es keine überfließende Knoten. Also ist der Preflow sogar ein Fluß.

Wegen Lemma F gibt es keinen augmentierenden Pfad. Also ist der Fluß gemäß Min-Cut-Max-Flow-Theorem maximal. □

Laufzeitanalyse

Lemma G

Es gibt von jedem überfließenden Knoten u einen Pfad nach s in G_f.

Beweis

Nehmen wir an, es gäbe keinen solchen Pfad.

Sei $U = \{\, v \mid v \text{ von } u \text{ in } G_f \text{ erreichbar} \,\}$.

$$\begin{aligned} 0 &\geq f(V-U,U) \text{ sonst Residualkante aus } U \text{ heraus} \\ &= f(V-U,U) + f(U,U) \\ &= f(V,U) = e(U) \end{aligned}$$

Also ist $e(u) = 0$. Widerspruch. □

Laufzeitanalyse

Lemma

Es werden nur $O(|V|^2)$ Lift-Operationen ausgeführt.

Beweis

- Eine Lift-Operation erhöht $h(u)$ um mindestens 1
- Sie wird nur auf überfließende Knoten angewendet
- $h(u) < 2|V|$, da Pfad in G_f zu s mit $h(s) = |V|$ (Lemma G)

Laufzeitanalyse

Lemma

Es werden nur $O(|V| \cdot |E|)$ saturierte Push-Operationen durchgeführt.

Beweis

- Wird eine Push-Operation auf (u, v) angewandt, dann führt (u, v) um 1 nach unten
- (u, v) verschwindet dann aus G_f (weil saturierter Push)
- Vor der nächsten Push-operation auf (u, v) muß ein Push auf (v, u) durchgeführt werden, dabei führt (u, v) um 1 nach oben
- Zwischen zwei saturierten Pushs auf (u, v) erhöht sich $h(u)$

$\Rightarrow$ Nur $O(|V|)$ saturierte Pushs auf (u, v) weil $h(u) = O(|V|)$.

Laufzeitanalyse

Lemma

Es werden nur $O(|V|^2 \cdot |E|)$ nicht-saturierte Push-Operationen durchgeführt.

Beweis

Sei $\Phi = \sum_{u \in X} h(u)$ mit $X = \{\, v \mid e(v) > 0 \,\}$.

- Eine Lift-Operation erhöht Φ höchstens um $|V|$
- Ein saturierter Push erhöht Φ höchstens um $2|V|$
- Ein nicht-saturierter Push erniedrigt Φ um mindestens 1

Insgesamt wird Φ höchstens um $|V|^2|E|$ erhöht.

$\Rightarrow$ höchstens $O(|V|^2|E|)$ nicht-saturierte Pushs.

Der einfache Preflow-Push-Algorithmus

Theorem

Ein maximaler Fluß kann in $O(|V|^2|E|)$ Schritten berechnet werden.

Beweisskizze

Der einfache Preflow-Push-Algorithmus muß so implementiert werden, daß

- in konstanter Zeit ein überfließender Knoten u gefunden wird, falls noch einer existiert,
- in konstanter Zeit bestimmt wird, ob eine Push- oder Lift-Operation anwendbar ist,
- eine Lift-Operation in $O(|V|)$ Schritten durchgeführt werden kann, und

- eine Push-Operation in konstanter Zeit durchgeführt wird.

Minimum Cost Flow Problem

Eingabe:

Ein s-t-Netzwerk $G = (V, E)$, eine Zahl $\bar{f}$ und eine *Kostenfunktion* $b\colon E \to \mathbf{R}_0^+$.

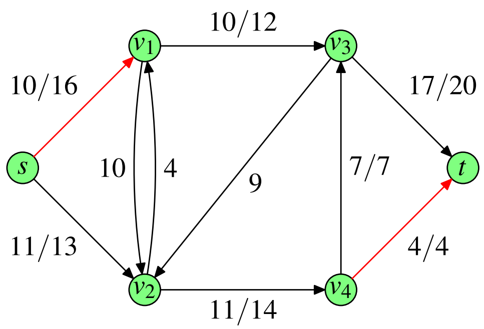
Gesucht:

Ein Fluß f mit $|f| = \bar{f}$ und minimalen Kosten, falls ein solcher existiert.

Die *Kosten* eines Flusses f sind

$$\sum_{e \in E} b(e) f(e).$$

Beispiel



Die schwarzen Kanten haben Kosten 1 und die roten haben Kosten 5.

Die Kosten des Flusses sind 126 und der Betrag 21.

Gibt es einen billigeren Fluß mit Betrag 21?

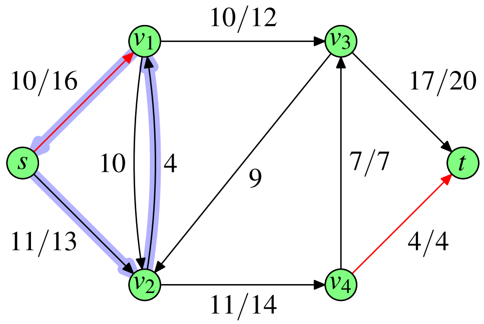
Minimum Cost Flow Problem

Theorem (Minimum Cost Flow Theorem)

1. Ein Fluß f mit Betrag $|f|$ hat genau dann minimale Kosten, wenn es in G_f keinen Kreis mit negativen Kosten gibt.
2. Seien f ein Fluß mit minimalen Kosten und p ein augmentierender Pfad mit minimalen Kosten. Dann entsteht wieder ein Fluß mit minimalen Kosten, wenn f entlang p augmentiert wird.

Die Kosten eines Pfads (oder Kreises) im Residualnetzwerk sind die Kosten der Kanten in Vorwärtsrichtung minus die Kosten der Kanten in Rückwärtsrichtung.

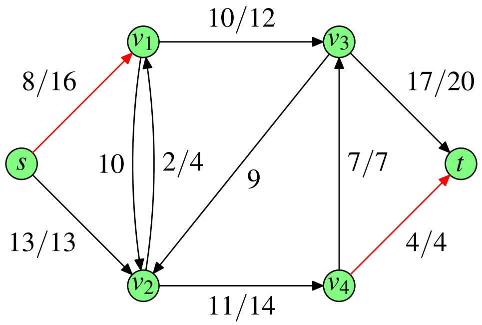
Beispiel



Da der Fluß nicht minimale Kosten hat, muß es einen Kreis in G_f mit negativen Kosten geben.

Seine Restkapazität ist 2 und seine Kosten sind -3.

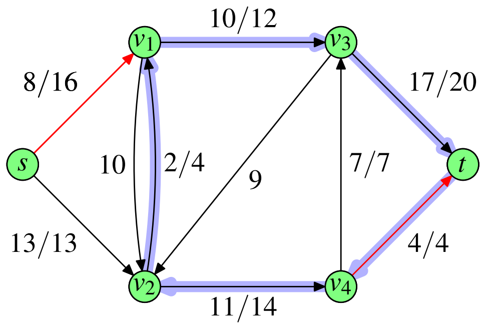
Beispiel



Augmentiert man den Fluß entlang des Kreises, erhalten wir einen neuen, billigeren Fluß.

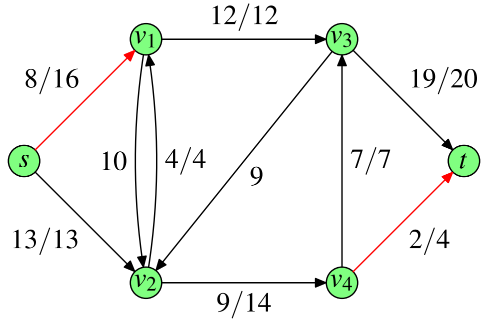
Die Kosten sind jetzt 120. Gibt es einen noch billigeren?

Beispiel



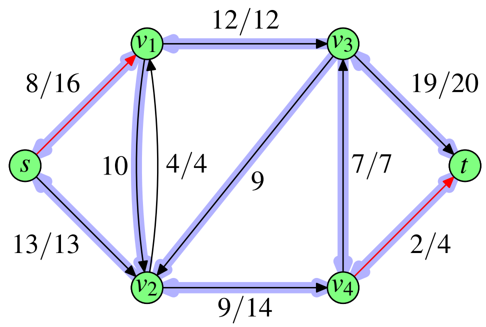
Da es wieder einen augmentierenden Kreis mit negativen Kosten gibt, kann der Fluß noch nicht minimale Kosten haben.

Beispiel



Hat der Fluß jetzt minimale Kosten?

Beispiel



Der Fluß hat minimale Kosten und es gibt keinen Kreis mit negativen Kosten im Residualnetzwerk.

Beweis (1. Teil)

Wenn ein Kreis mit negativen Kosten existiert, dann hat der Fluß nicht minimale Kosten:

Sei K ein Kreis in G_f mit negativen Kosten und Restkapazität $c_f(K)$.

Ändern wir den Fluß indem wir ihn auf jeder Kante von K um $c_f(K)$ erhöhen, dann sinken die Kosten.

$\Rightarrow$ Dies widerspricht der Annahme, daß f minimale Kosten hat.

Andere Richtung:

Angenommen f hat nicht minimale Kosten. Dann existiert ein Fluß f^* mit kleineren Kosten, aber $|f| = |f^*|$.

Sei $\Delta = f^* - f$. Da $|f| = |f^*|$, gilt $\Delta(V, s) = \Delta(V, t) = 0$.

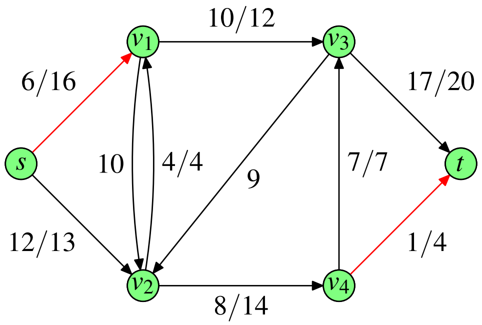
Es gilt also $\Delta(V, u) = 0$ für alle $u \in V$.

$\Rightarrow$ $\Delta(u, v) > 0$ ist eine Vereinigung von Flüssen auf Kreisen.

$\Rightarrow$ Einer der Kreise muß negative Kosten haben

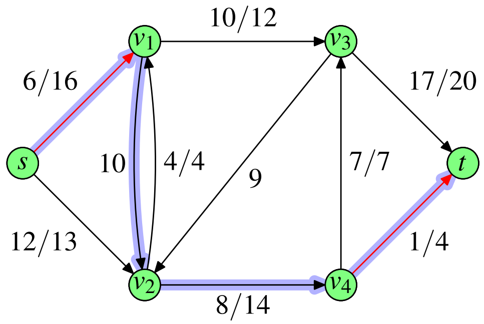
(Damit ist der erste Teil bewiesen.)

Beispiel



Der Fluß hat minimale Kosten. Was passiert, wenn er mit einem augmentierenden Pfad augmentiert wird?

Beispiel



Dieser augmentierende Pfad hat nicht minimale Kosten, denn von s nach v_2 gibt es eine billigere Abkürzung.

Geht man die Abkürzung und dann den übersprungenen Teil rückwärts, erhalten wir einen Kreis mit negativen Kosten.

Beweis (2. Teil)

Damit durch eine Augmentierung ein Kreis im Residualnetzwerk entsteht, muß der augmentierende Pfad durch eine Kante des Kreises rückwärts gehen.

Sei p ein augmentierender Pfad, durch den ein Kreis K mit negativen Kosten entsteht. Die Kante (u, v) liege auf p und die Kante (v, u) liege auf K.

Sei p' folgender Pfad: Von s entlang p bis u, dann entlang K bis v, dann entlang p bis t.

Die Kosten von p' sind dann kleiner als von p. Also kann p kein augmentierender Pfad mit minimalen Kosten sein.

Ein einfacher Algorithmus

Gesucht: Fluß mit Betrag B und minimalen Kosten

Initialisiere Fluß f zu 0
while es gibt einen augmentierenden Pfad p **do**
    finde einen augmentierenden Pfad p
        mit minimalen Kosten
    augmentiere f entlang p mit $\min\{B - |f|, c_f(p)\}$
return f

Durch das Minimum Cost Flow Theorem ist garantiert, daß der gefundene Fluß minimale Kosten hat.

Die Edmonds–Karp–Variante

```
for each edge (u,v) ∈ E do
   f(u,v) := 0
   f(v,u) := 0
while there exists a path from s to t in G_f do
   p := a shortest, minimum cost path from s to t in G_f
   c_f(p) := min{ c_f(u,v) | (u,v) is in p }
   for each edge (u,v) in p do
      f(u,v) := f(u,v) + min{B − |f|, c_f(p)}
      f(v,u) := −f(u,v)
return f
```


Lemma D'

Gegeben ist ein s-t-Netzwerk $G = (V, E)$. Sei f ein Fluß, der während der Ausführung der Edmonds–Karp–Variante vorkommt und f' der Fluß nach der anschließenden Augmentierung.

Dann gilt: $\delta'(s, v) \geq \delta(s, v)$ für alle $v \in V$.

Hierbei sind $\delta(u, v)$ (bzw. $\delta'(u, v)$) die Kosten des billigsten Pfads von u nach v in G_f (bzw. in $G_{f'}$).

Beweis:

Wie Lemma D

Theorem

Die Edmonds–Karp–Variante findet einen Fluß mit Betrag B und minimalen Kosten in endlich vielen Schritten.

Beweis

- Solange die Kosten der augmentierenden Pfade gleichbleiben, werden unter ihnen kürzeste gewählt und der Algorithmus arbeitet wie der Edmonds–Karp–Algorithmus.

  $\Rightarrow$ Nach $O(|E| \cdot |V|)$ Iterationen, wird ein Pfad mit höheren Kosten gewählt (Lemma D').

- Es gibt nur endlich viele, mögliche augmentierende Pfade und damit nur endlich viele mögliche Kosten.

Bipartites Matching maximalen Gewichts

Gegeben:

Ein bipartiter, ungerichteter Graph (V_1, V_2, E) und eine Gewichtsfunktion $b\colon E \to \mathbf{R}_0^+$.

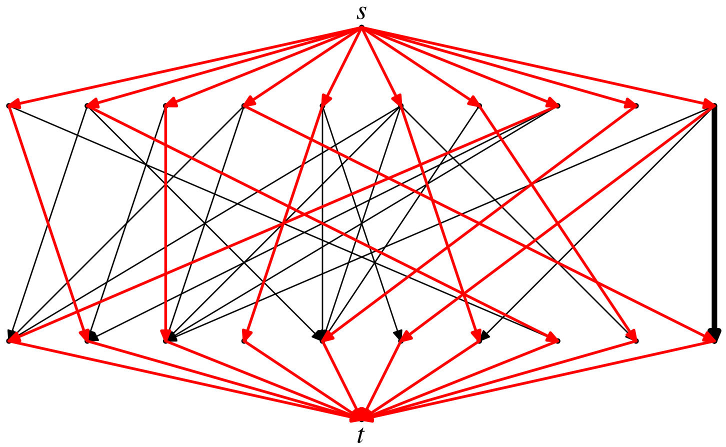
Gesucht:

Ein Matching mit maximalem Gewicht.

Das Gewicht eines Matchings M ist

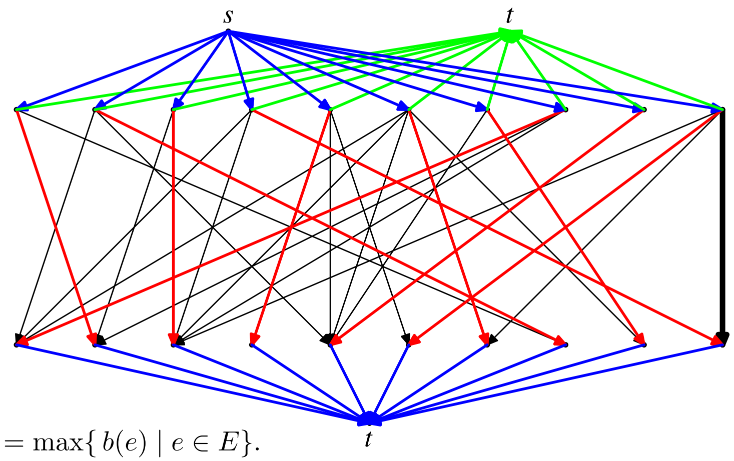
$$b(M) = \sum_{e \in M} b(e).$$

Beispiel



Alle Kanten haben Gewicht 1, nur die dicke Kante hat 3.

Hat das perfekte Matching auch maximales Gewicht?

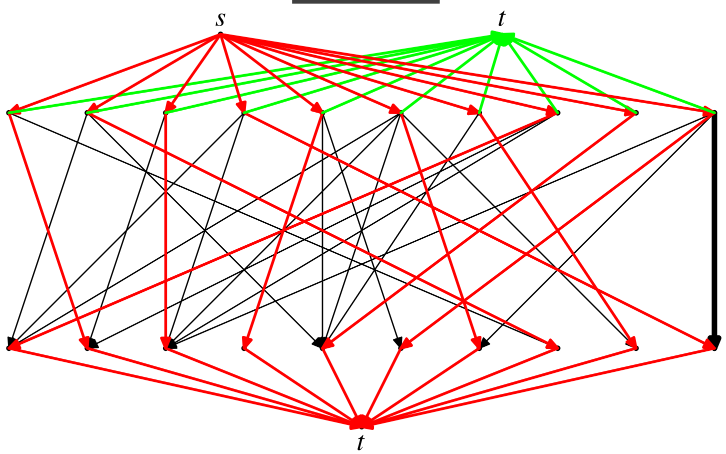


$w^* = \max\{\, b(e) \mid e \in E \}$.

Blaue Kanten haben Kosten 0, grüne Kanten haben Kosten w^*, eine Matching-Kante e hat Kosten $w^* - b(e)$.

Matching mit Gewicht $b \Longrightarrow$ Fluß mit Kosten $nw^* - b$.

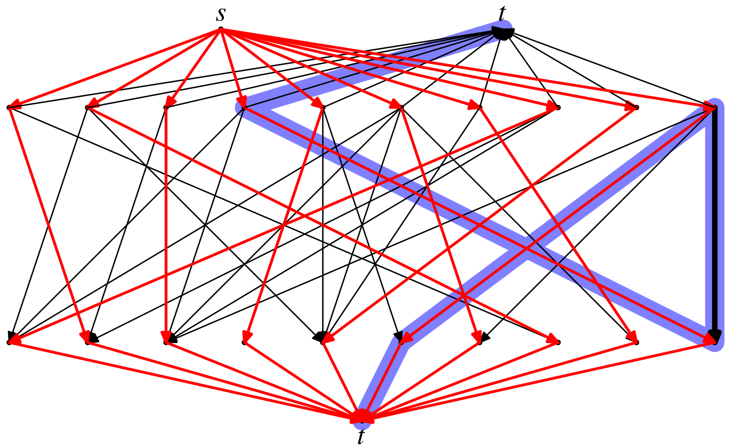
Beispiel



Dem perfekten Matching entspricht der rote Fluß mit Betrag 10.

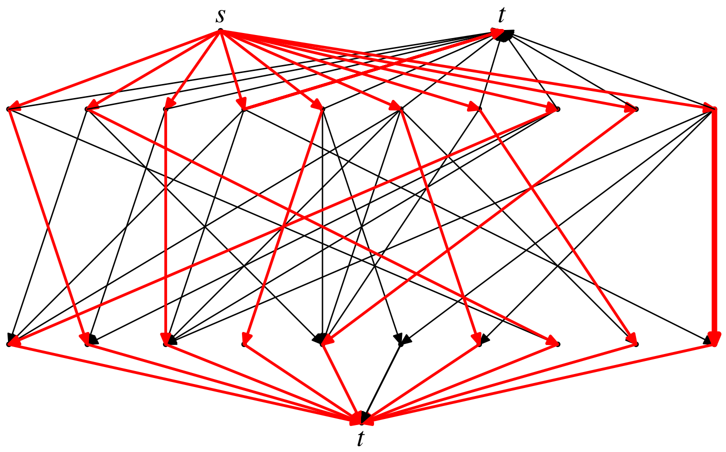
Wie groß sind die Kosten? ($w^* = 3$)

Beispiel



Die Kosten sind nicht minimal, denn es gibt einen Kreis in G_f mit negativen Kosten.

Beispiel



Jetzt hat der Fluß Betrag 10 und minimale Kosten. Das entsprechende Matching ist kleiner, hat aber maximales Gewicht.

Bipartites Matching maximalen Gewichts

Bipartites Matching maximalen Gewichts läßt sich mittels Flusses minimaler Kosten lösen:

Die Kosten des Flusses sind

- $(n - |M|)w^*$ und
- $\sum_{e \in M} (w^* - b(e))$

Zusammen sind es $nw^* - b(M)$ (mit $n = |V_1| \geq |V_2|$)

Dieser Wert ist minimal, wenn $b(M)$ maximal ist.□

Lineares Programmieren

Eingabe:

Eine Menge von linearen Ungleichungen $S_i\colon \sum_{j=1}^{m} A_{ij}x_j \leq b_i$.

Eine lineare Optimierungsfunktion $f(x) = \sum_{j=1}^{m} c_j x_j$.

Problem:

Gesucht ist eine Belegung $x \geq 0$, für die alle Ungleichungen erfüllt sind und dir $f(x)$ maximiert.

Beispiel

Gegeben ein s-t-Netzwerk $G = (V, E)$.

Maximiere $\sum_{v \in V} f(s, v)$ unter den Bedingungen

- $f(u, v) = -f(u, v)$ für alle $(u, v) \in E$,
- $\sum_{u \in V} f(v, u) = 0$ für alle $u \in V - \{s, t\}$,
- $f(u, v) \leq c(u, v)$.

⇒ Lineares Programm für das Maximale-Fluß-Problem.

Modellierung als Lineares Programm

Maximiere $c^T x$ unter $Ax \leq b$ und $x \geq 0$.

Dabei ist $x \in \mathbf{R}^n$, $c \in \mathbf{R}^n$, $b \in \mathbf{R}^m$ und $A \in \mathbf{R}^{m \times n}$.

Viele abweichende Probleme lassen sich in diese Form bringen:

- Minimiere $c^T x \quad \mapsto \quad$ Maximiere $-c^T x$
- $a^T x \geq b \quad \mapsto \quad -a^T x \leq -b$
- Unbeschränktes $x_i \quad \mapsto \quad x_i' - x_i''$ mit $x_i', x_i'' \geq 0$

Beispiel: Wassermischen

Wir brauchen 100 Liter Wasser mit 60°C und nehmen

- x_1 Liter mit 20°C zu 1 DM/l
- x_2 Liter mit 50°C zu 2 DM/l
- x_3 Liter mit 100°C zu 3 DM/l

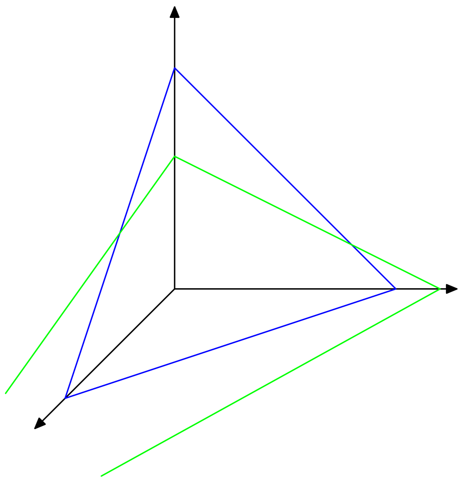
Minimiere $x_1 + 2x_2 + 3x_3$ (die Kosten) unter

- $x_1 + x_2 + x_3 = 100$
- $20x_1 + 50x_2 + 100x_3 = 6000$

Beispiel: Wassermischen

$x_1 + x_2 + x_3 = 100$

$20x_1 + 50x_2 + 100x_3 = 6000$



Kanonische und Normalform

Kanonische Form: Minimiere $c^T x$ unter $Ax \geq b$

Normalform: Minimiere $c^T x$ unter $Ax = b$, $x \geq 0$

Umwandlung Kanonisch $\rightarrow$ Normalform:

Minimiere $c_T x^+ - c^T x^-$ unter $Ax^+ - Ax^- - Is = b$, $x^+, x^-, s \geq 0$

Umwandlung Normalform $\rightarrow$ Kanonisch:

Minimiere $c^T x$ unter $\begin{pmatrix} A \\ -A \\ I \end{pmatrix} x \geq \begin{pmatrix} b \\ -b \\ 0 \end{pmatrix}$

Beispiel

$$\begin{array}{rcrcrcr} x_1 & + & x_2 & + & x_3 & = & 100 \\ 20x_1 & + & 50x_2 & + & 100x_3 & = & 6000 \\ & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

$$\begin{array}{rrcrcrcr} & x_1 & + & x_2 & + & x_3 & \geq & 100 \\ - & x_1 & - & x_2 & - & x_3 & \geq & -100 \\ & 20x_1 & + & 50x_2 & + & 100x_3 & \geq & 6000 \\ - & 20x_1 & - & 50x_2 & - & 100x_3 & \geq & -6000 \\ & x_1 & & & & & \geq & 0 \\ & & & x_2 & & & \geq & 0 \\ & & & & & x_3 & \geq & 0 \end{array}$$

Beispiel

$$\begin{array}{rcrcr} x_1 & + & x_2 & \geq & 5 \\ x_1 & - & 5x_2 & \geq & 6 \\ 3x_1 & - & x_2 & \geq & -2 \end{array}$$

$$\begin{array}{rcrcrcrcrl} x_1 & + & x_2 & - & x_3 & & & & & = 5 \\ x_1 & - & 5x_2 & & & - & x_4 & & & = 5 \\ 3x_1 & - & x_2 & & & & & - & x_5 & = 5 \\ & & & & & & & x_3, x_4, & x_5 & \geq 0 \end{array}$$

x_3, x_4, x_5 sind *Schlupfvariablen.*

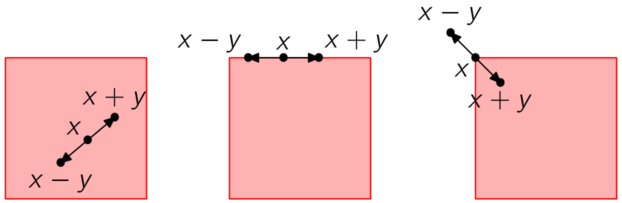
Ist dies bereits Normalform?

Geometrie des Linearen Programmierens

Sei $P = \{\, x \mid Ax = b,\ x \geq 0 \,\} \subseteq \mathbf{R}^n$.

Definition

x ist eine *Ecke* von P, wenn es kein $y \neq 0$ mit $x + y \in P$ und $x - y \in P$ gibt.



keine Ecke keine Ecke Ecke!

Geometrie des Linearen Programmierens

Theorem A

Sei $P = \{\, x \mid Ax = b,\ x \geq 0\}$.

$\min\{\, c^T x \mid x \in P\}$ sei endlich (d.h., es existiere).

Dann gibt es für jedes $x \in P$ eine Ecke $x' \in P$ mit

$$c^T x' \leq c^T x.$$

Korollar

Das Minimum wird (auch) in einer Ecke angenommen.

Beweis

Sei $P = \{ x \mid Ax = b,\, x \geq 0\}$.

Wenn $x \in P$ bereits eine Ecke ist, dann $x' := x$.

Anderenfalls gibt es ein $y \neq 0$ mit $x + y, x - y \in P$.

Es gilt $Ay = 0$, denn $A(x + y) = A(x - y) = b$.

O.B.d.A. sei $c^T y \leq 0$ (sonst nehme $-y$).

Falls $c^T y = 0$, existiere ein j mit $y_j < 0$ (sonst nehme $-y$).

Beweis (Fortsetzung)

Wir zeigen nun, daß $y \geq 0$ nicht gelten kann:

Wenn doch, dann $c^T y < 0$.

Dann ist $x + \lambda y \in P$ für alle $\lambda \geq 0$, denn $A(x + \lambda y)b$ wegen $Ay = 0$.

Aber: $c_T(x + \lambda y) = cTx + \lambda c^T y \to -\infty$ für $\lambda \to \infty$.

Also ist $\min\{ c^T x \mid x \in P\}$ nicht endlich. Widerspruch.

Beweis (Fortsetzung)

Es gibt ein j mit $y_j < 0$.

- Wähle $\lambda = \min\left\{ -\frac{x_j}{y_j} \;\middle|\; y_j < 0 \right\} = -\frac{x_k}{y_k}$.
- Dies ist das größte λ mit $x + \lambda y \geq 0$.
- $x + \lambda y \in P$, denn $Ay = 0 \Rightarrow A(x + \lambda y) = b$.
- $(x + \lambda y)_k = 0$, aber $x_k > 0$.

Ersetze jetzt x durch $x + \lambda y$.

Da es nur n Komponenten gibt, kann nach höchstens n Iterationen kein weiteres y mehr existieren und x muß eine Ecke sein. □

Ecken und Basen

Theorem B

Sei $P = \{ x \mid Ax \geq b,\, x \geq 0 \}$ und $x \in P$. Mit A_x bezeichnen wir die Untermatrix von A, die aus den Spalten j besteht, für die $x_j > 0$ gilt.

Dann ist x genau dann eine Ecke von P, wenn die Spalten von A_x linear unabhängig sind.

Beispiel

$$A = \begin{pmatrix} 2 & 1 & 3 & 0 \\ 7 & 3 & 2 & 1 \\ 0 & 0 & 0 & 5 \end{pmatrix} \qquad x = \begin{pmatrix} 2 \\ 0 \\ 1 \\ 0 \end{pmatrix} \qquad A_x = \begin{pmatrix} 2 & 3 \\ 7 & 2 \\ 0 & 0 \end{pmatrix}$$

Korollar Es gibt nur endlich viele Ecken.

Beweis

„$\Longleftarrow$“

Nehmen wir an, x sei keine Ecke. Es gibt also ein $y \neq 0$ mit $x + y, x - y \in P$. Sei A_y die Untermatrix, die zu den nicht-Null-Komponenten von y gehört.

- Aus $A(x + y) = A(x - y) = b$ folgt wieder $Ay = 0$.
  Wegen $y \neq 0$ hat A_y linear abhängige Spalten.
- $x_j = 0 \Rightarrow y_j = 0$, denn $x + y \geq 0$ und $x - y \geq 0$.
  Daher ist A_y eine Untermatrix von A_x.

Dann hat auch A_x linear abhängige Spalten.

Beweis (Fortsetzung)

„$\Longrightarrow$“

Nehmen wir an, A_x hat linear abhängige Spalten. Dann gibt es ein $y \neq 0$ mit $A_x y = 0$.

Durch Erweiterung mit 0-Komponenten erhalten wir ein $y \neq 0$ mit $Ay = 0$ und

$$x_j = 0 \Rightarrow y_j = 0.$$

Für ein genügend kleines $\epsilon > 0$ muß dann $x + \epsilon y \in p$ und $x - \epsilon y \in P$ gelten.

Also ist x keine Ecke! $\square$

Basen

Sei $P = \{\, x \mid Ax \geq b,\ x \geq 0 \,\}$, $A \in \mathbf{R}^{m \times n}$.

Für $B \subseteq \{1, \ldots, n\}$ sei A_B die Untermatrix von A, die aus allen Spalten in B besteht.

1. Falls $B = \{\, j \mid x_j > 0 \,\}$ für ein $x \in P$, dann $A_B = A_x$.
2. Falls x eine Ecke von P ist, dann ist A_x eine Matrix mit vollem Spaltenrang $\leq m$.
3. Wir nehmen ab jetzt an, daß $\mathrm{Rang}(A) = m$ gilt, denn sonst gibt es eine redundante Gleichung, die eliminiert werden kann.
4. A_x läßt sich immer zu einer quadratischen Matrix A_B mit m linear unabhängigen Spalten erweitern.
5. Eine quadratische Matrix A_B mit vollem Rang heißt *Basis*.

Basen

Beispiel

$$A = \begin{pmatrix} 2 & 1 & 3 & 0 \\ 7 & 3 & 2 & 1 \\ 0 & 0 & 0 & 5 \end{pmatrix} \quad x = \begin{pmatrix} 2 \\ 0 \\ 1 \\ 0 \end{pmatrix} \quad A_x = \begin{pmatrix} 2 & 3 \\ 7 & 2 \\ 0 & 0 \end{pmatrix}$$

$$A_B = \begin{pmatrix} 2 & 3 & 0 \\ 7 & 2 & 1 \\ 0 & 0 & 5 \end{pmatrix} \text{ mit } B = \{1, 3, 4\} \text{ ist eine Basis}$$

Sei A_B eine Basis und $N = \{1, \ldots, n\} - B$.

Die Variablen in B heißen *Basisvariablen* und die Variablen in N heißen *Nicht-Basisvariablen*.

Ax kann als $A_B x_B + A_N x_N$ geschrieben werden und daher

$$x_B = A_B^{-1} b - A_B^{-1} A_N x_N$$

für $x \in P$ (denn $A_B x_B + A_N x_N = b$).

Wenn x eine Ecke zu A_B ist, dann ist $A_B x_B = b$ wegen $x_N = 0$.

Die Zielfunktion läßt sich schreiben als:

$$\begin{aligned} c^T x &= c_B^T x_B + c_N^T x_N \\ &= c_B^T (A_B^{-1} b - A_B^{-1} A_N x_N) + c_N^T x_N \\ &= c_B^T A_B^{-1} b + \underbrace{(c_N^T - c_B^T A_B^{-1} A_N)}_{\bar{c}^T} x_N \end{aligned}$$

Der Simplex-Algorithmus

$$c^T x = c_B^T A_B^{-1} b + \underbrace{(c_N^T - c_B^T A_B^{-1} A_N)}_{\bar{c}^T} x_N$$

Falls $\bar{c} \geq 0$, dann ist $x \in P$ eine optimale Lösung.

Sei $x_B = A_B^{-1} b$ und $x_N = 0$ die zu A_B gehörige Ecke. Insbesondere gelte $x \in P$, also $x_B \geq 0$.

Falls $x_j > 0$ für alle $j \in B$, dann ist A_B *nicht entartet*, andernfalls *entartet*. Wir nehmen zunächst an, alle Basen seien nicht entartet.

> Wähle $j \in N$ mit $\bar{c}_j < 0$
> $x_j := x_j + \lambda$, $x_B := A_B^{-1} b - A_B^{-1} A_N x_N$
> für ein maximales λ, so daß $x_B \geq 0$
> Wähle $k \in B$ mit $x_k = 0$
> $B := B \cup \{j\} - \{k\}$, $N := \{1, \ldots, n\} - B$

Der Simplex-Algorithmus

> Wähle $j \in N$ mit $\bar{c}_j < 0$
> $x_j := x_j + \lambda$, $x_B := A_B^{-1} b - A_B^{-1} A_N x_N$
> für ein maximales λ, so daß $x_B \geq 0$
> Wähle $k \in B$ mit $x_k = 0$
> $B := B \cup \{j\} - \{k\}$, $N := \{1, \ldots, n\} - B$

Wir erhalten eine neue Basis A_B mit einem neuen Vektor x und einer kleineren Zielfunktion $c^T x$:

$$c^T x = c_B^T A_B^{-1} b + \bar{c}^T x_N$$

Da es nur endlich viele Basen gibt, kann man durch Wiederholen in endlicher Zeit eine optimale Lösung finden.

Beispiel

Minimiere $c^T x$ unter $Ax = b$ und $x \geq 0$ mit

$$A = \begin{pmatrix} 2 & 1 & 3 & 0 \\ 7 & 3 & 2 & 1 \\ 0 & 0 & 0 & 5 \end{pmatrix} \qquad b = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \qquad c = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Wir wählen folgende Basis A_B:

$$A_B = \begin{pmatrix} 2 & 3 & 0 \\ 7 & 2 & 1 \\ 0 & 0 & 5 \end{pmatrix} \qquad A_B^{-1} = \frac{1}{85} \begin{pmatrix} -10 & 15 & -3 \\ 35 & -10 & 2 \\ 0 & 0 & 17 \end{pmatrix} \qquad x_B = \frac{1}{85} \begin{pmatrix} 11 \\ 21 \\ 51 \end{pmatrix}$$

Beispiel

$$A_B = \begin{pmatrix} 2 & 3 & 0 \\ 7 & 2 & 1 \\ 0 & 0 & 5 \end{pmatrix} \quad A_B^{-1} = \frac{1}{85}\begin{pmatrix} -10 & 15 & -3 \\ 35 & -10 & 2 \\ 0 & 0 & 17 \end{pmatrix} \quad x_B = \frac{1}{85}\begin{pmatrix} 11 \\ 21 \\ 51 \end{pmatrix}$$

$$\bar{c}^T = c_N^T - c_B^T A_B^{-1} A_N =$$

$$1 - \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \frac{1}{85} \begin{pmatrix} -10 & 15 & -3 \\ 35 & -10 & 2 \\ 0 & 0 & 17 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 0 \end{pmatrix} = \frac{9}{17}$$

Da $\bar{c} \geq 0$ ist $x = \frac{1}{85}\begin{pmatrix} 11 & 21 & 51 & 0 \end{pmatrix}^T$ optimal.

Pivotieren

Falls A_B entartet ist, kann es ein Problem geben: Obwohl $\bar{c}_j < 0$, können wir x_j nicht erhöhen, ohne eine Komponente von x_B negativ zu machen.

Dies kommt in der Praxis oft vor.

Wir können dann *pivotieren*: Wir wählen ein $k \in B$ mit $x_k = 0$ und ein $j \in N$. Dann setzen wir

$$B := B - \{k\} \cup \{j\} \text{ und } N := N - \{j\} \cup \{k\}.$$

Dabei ändert sich x und $c^T x$ nicht, aber wir haben eine andere Basis.

Es kann zu einem Zyklus kommen und der Algorithmus terminiert nicht.

Das kommt in der Praxis nicht oft vor.

Maximiere z unter den Bedingungen:

$$\begin{array}{rcrcrcrl}
z & - & 300x_1 & - & 500x_2 & = & -36000 & (\alpha) \\
x_3 & + & x_1 & + & 2x_2 & = & 170 & (\beta) \\
x_4 & + & x_1 & + & x_2 & = & 150 & (\gamma) \\
x_5 & & & + & 3x_2 & = & 180 & (\delta) \\
 & & & & x_1, x_2, x_3, x_4, x_5 & \geq & 0 &
\end{array}$$

$$\begin{array}{rcrcrcrl}
z & - & 300x_1 & + & \frac{500}{3}x_5 & = & -6000 & (\alpha' = \alpha + 500\delta') \\
x_3 & + & x_1 & - & \frac{2}{3}x_5 & = & 50 & (\beta' = \beta - 2\delta') \\
x_4 & + & x_1 & - & \frac{1}{3}x_5 & = & 90 & (\gamma' = \gamma - \delta') \\
x_2 & & & + & \frac{1}{3}x_5 & = & 60 & (\delta' = \frac{1}{3}\delta) \\
 & & & & x_1, x_2, x_3, x_4, x_5 & \geq & 0 &
\end{array}$$

$$\begin{array}{rcrcrcrl}
z & - & 300x_1 & + & \frac{500}{3}x_5 & = & -6000 & (\alpha') \\
x_3 & + & x_1 & - & \frac{2}{3}x_5 & = & 50 & (\beta') \\
x_4 & + & x_1 & - & \frac{1}{3}x_5 & = & 90 & (\gamma') \\
x_2 & & & + & \frac{1}{3}x_5 & = & 60 & (\delta') \\
 & & & & x_1, x_2, x_3, x_4, x_5 & \geq & 0 &
\end{array}$$

$$\begin{array}{rcrcrcrl}
z & + & 300x_3 & - & \frac{100}{3}x_5 & = & 9000 & (\alpha'' = \alpha' + 300\beta'') \\
x_1 & + & x_3 & - & \frac{2}{3}x_5 & = & 50 & (\beta'' = \beta') \\
x_4 & - & x_3 & + & \frac{1}{3}x_5 & = & 40 & (\gamma'' = \gamma' - \beta'') \\
x_2 & & & + & \frac{1}{3}x_5 & = & 60 & (\delta'' = \delta' + 0\beta'') \\
 & & & & x_1, x_2, x_3, x_4, x_5 & \geq & 0 &
\end{array}$$

$$
\begin{array}{ccccccccl}
z & + & 300x_3 & - & \frac{100}{3}x_5 & = & 9000 & (\alpha'') \\
x_1 & + & x_3 & - & \frac{2}{3}x_5 & = & 50 & (\beta'') \\
x_4 & - & x_3 & + & \frac{1}{3}x_5 & = & 40 & (\gamma'') \\
x_2 & & & + & \frac{1}{3}x_5 & = & 60 & (\delta'') \\
 & & & & x_1, x_2, x_3, x_4, x_5 & \geq & 0 &
\end{array}
$$

$$
\begin{array}{ccccccccl}
z & + & 200x_3 & + & 100x_4 & = & 13000 & (\alpha''' = \alpha'' + \frac{100}{3}\gamma''') \\
x_1 & - & x_3 & + & 2x_4 & = & 130 & (\beta''' = \beta''\frac{2}{3}\gamma''') \\
x_5 & - & 3x_3 & + & 3x_4 & = & 120 & (\gamma''' = 3\gamma'') \\
x_2 & + & x_3 & - & x_4 & = & 20 & (\delta''' = \delta'' - \frac{1}{3}\gamma''') \\
 & & & & x_1, x_2, x_3, x_4, x_5 & \geq & 0 &
\end{array}
$$

Das Simplex-Tableau

1. Maximiere $z = c^T x$ unter $Ax = b$.
2. Wir beginnen mit einer *zulässigen Basislösung*. Wenn A_B eine Basis ist, dann ist $x_B = A_B^{-1}, x_N = 0$ eine *Basislösung*. Wenn $x_B \geq 0$, dann ist x eine *zulässige Basislösung*.
3. Für jedes $x_N \in \mathbf{R}^{n-m}$ gibt es genau ein $x_B \in \mathbf{R}^m$, so daß $Ax = b$, nämlich $x_B = A_B^{-1}b - A_B^{-1}A_N x_N$.
4. Ebenso läßt sich $z = c^T x$ als $z = \bar{c}^T x_N + c_B^T A_B^{-1} b$ schreiben, falls $Ax = b$.
5. Im *Simplex-Tableau* werden z und x_B als (nicht homogene) Linearkombinationen von x_N dargestellt.

Das Simplex-Tableau

	y_1	$\cdots$	y_k	
z	$\bar{a}_{01}$	$\cdots$	$\bar{a}_{0k}$	$\bar{b}_0$
x_1	$\bar{a}_{11}$	$\cdots$	$\bar{a}_{1k}$	$\bar{b}_1$
x_2	$\bar{a}_{21}$	$\cdots$	$\bar{a}_{2k}$	$\bar{b}_2$
$\vdots$	$\vdots$		$\vdots$	$\vdots$
x_n	$\bar{a}_{21}$	$\cdots$	$\bar{a}_{2k}$	$\bar{b}_n$

Die Basisvariablen sind x_i, die Nicht-Basisvariablen y_i. Das Tableau wird so gelesen:

$$z + \sum_{j=1}^{k} \bar{a}_{0j} y_j = \bar{b}_0 \qquad x_i + \sum_{j=1}^{k} \bar{a}_{ij} y_j = \bar{b}_i$$

Das Simplex-Tableau

	y_1	$\cdots$	y_k	
z	$\bar{a}_{01}$	$\cdots$	$\bar{a}_{0k}$	$\bar{b}_0$
x_1	$\bar{a}_{11}$	$\cdots$	$\bar{a}_{1k}$	$\bar{b}_1$
x_2	$\bar{a}_{21}$	$\cdots$	$\bar{a}_{2k}$	$\bar{b}_2$
$\vdots$	$\vdots$		$\vdots$	$\vdots$
x_n	$\bar{a}_{21}$	$\cdots$	$\bar{a}_{2k}$	$\bar{b}_n$

Wähle $s \in \{1, \ldots, k\}$ mit $\bar{a}_{0s} < 0$ (Pivotspalte).

Wähle $t \in \{1, \ldots, n\}$ mit minimalem $\bar{b}_t / \bar{a}_{ts}$ (Pivotzeile).

$\bar{a}_{tj} := \bar{a}_{tj} / \bar{a}_{ts}$ für $j \neq s$, $\bar{a}_{ts} := 1/\bar{a}_{ts}$, $\bar{b}_t := \bar{b}_t / \bar{a}_{ts}$.

$\bar{a}_{ij} := \bar{a}_{ij} - \bar{a}_{tj}\bar{a}_{is}$ für $j \neq s$, $\bar{a}_{is} := -\bar{a}_{ts}\bar{a}_{is}$, $\bar{b}_i := \bar{b}_i - \bar{b}_t \bar{a}_{is}$ $(i \neq t)$.

Vertausche x_t mit y_s.

Beispiel

	x_1	x_2	
z	-300	-500	-36000
x_3	1	2	170
x_4	1	1	150
x_5	0	3	180

	x_1	x_5	
z	-300	$500/3$	-6000
x_3	1	$-2/3$	50
x_4	1	$-1/3$	90
x_2	0	$1/3$	60

Beispiel

	x_1	x_5	
z	-300	$500/3$	-6000
x_3	1	$-2/3$	50
x_4	1	$-1/3$	90
x_2	0	$1/3$	60

	x_3	x_5	
z	300	$-100/3$	9000
x_1	1	$-2/3$	50
x_4	-1	$1/3$	40
x_2	0	$1/3$	60

Beispiel

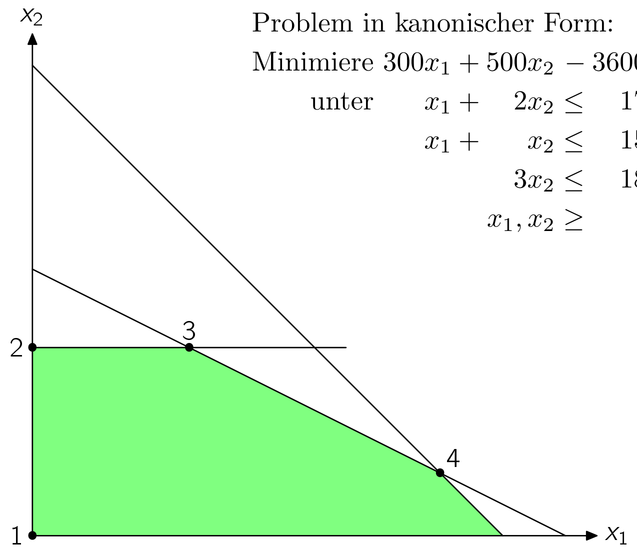
	x_3	x_5	
z	300	$-100/3$	9000
x_1	1	$-2/3$	50
x_4	-1	$1/3$	40
x_2	0	$1/3$	60

	x_3	x_4	
z	300	100	13000
x_1	-1	2	130
x_5	-3	3	120
x_2	1	-1	20

Beispiel

Problem in kanonischer Form:

$$
\begin{aligned}
\text{Minimiere } & 300x_1 + 500x_2 - 36000 \\
\text{unter } & x_1 + 2x_2 \le 170 \\
& x_1 + x_2 \le 150 \\
& 3x_2 \le 180 \\
& x_1, x_2 \ge 0
\end{aligned}
$$



Beispiel

	x_1	x_2	
z	-300	-500	-36000
x_3	1	2	170
x_4	1	1	150
x_5	0	3	180

	x_4	x_2	
z	300	-200	9000
x_3	-1	1	20
x_1	1	1	150
x_5	0	3	180

Beispiel

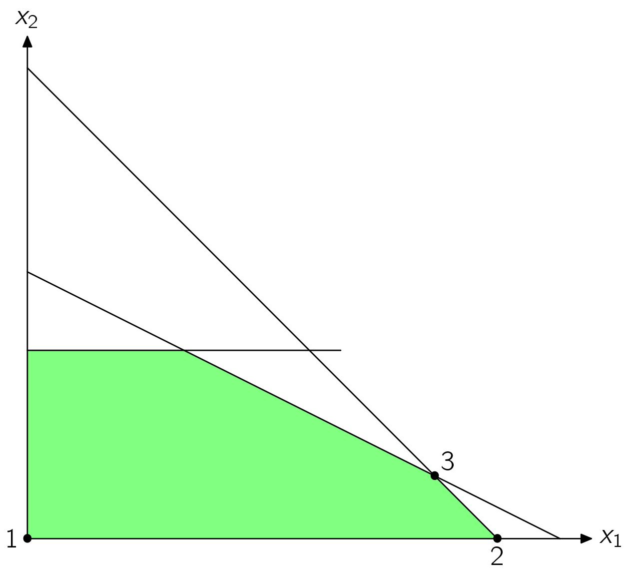
	x_4	x_2	
z	300	-200	9000
x_3	-1	1	20
x_1	1	1	150
x_5	0	3	180

	x_4	x_3	
z	100	200	13000
x_2	-1	1	20
x_1	2	-1	130
x_5	3	-3	120

Beispiel

Problem in kanonischer Form:

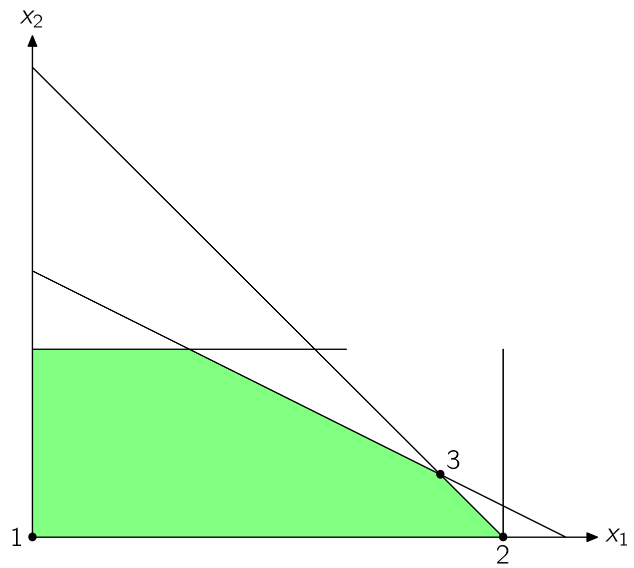
$$
\begin{array}{lrrl}
\text{Minimiere } 300x_1 + 500x_2 - 36000 & & & \\
\text{unter} \quad x_1 + & 2x_2 \leq & 170 \\
x_1 + & x_2 \leq & 150 \\
& 3x_2 \leq & 180 \\
& x_1, x_2 \geq & 0
\end{array}
$$



Entartete Basen

Problem in kanonischer Form:

$$
\begin{array}{lrrrr}
\text{Minimiere } 300x_1 + 500x_2 - 36000 & & & & \\
\text{unter} & x_1 + & 2x_2 & \le & 170 \\
 & x_1 + & x_2 & \le & 150 \\
 & & 3x_2 & \le & 180 \\
 & x_1 & & \le & 150 \\
 & & x_1, x_2 & \ge & 0
\end{array}
$$



Entartete Basen

	x_1	x_2	
z	-300	-500	-36000
x_3	1	2	170
x_4	1	1	150
x_5	0	3	180
x_6	1	0	150

	x_4	x_2	
z	300	-200	9000
x_3	-1	1	20
x_1	1	1	150
x_5	0	3	180
x_6	-1	-1	**0**

Entartete Basen

	x_4	x_2	
z	300	-200	9000
x_3	-1	1	20
x_1	1	1	150
x_5	0	3	180
x_6	-1	-1	**0**

	x_4	x_3	
z	100	200	13000
x_2	-1	1	20
x_1	2	-1	130
x_5	3	-3	120
x_6	-2	1	20

Entartete Basen

	x_1	x_2	
z	-300	-500	-36000
x_3	1	2	170
x_4	1	1	150
x_5	0	3	180
x_6	1	0	150

	x_6	x_2	
z	300	-500	9000
x_3	-1	2	20
x_4	-1	1	**0**
x_5	0	3	180
x_1	1	0	150

Entartete Basen

	x_6	x_2	
z	300	-500	9000
x_3	-1	2	20
x_4	-1	1	**0**
x_5	0	3	180
x_1	1	0	150

	x_6	x_4	
z	-200	500	9000
x_3	1	2	20
x_2	-1	1	**0**
x_5	3	-3	180
x_1	1	0	150

Entartete Basen

	x_6	x_4	
z	-200	500	9000
x_3	1	2	20
x_2	-1	1	**0**
x_5	3	-3	180
x_1	1	0	150

	x_3	x_4	
z	200	900	13000
x_6	1	2	20
x_2	1	3	20
x_5	-3	-9	120
x_1	-1	-2	130

Wahl des Pivot-Elements

Zusammenfassung:

- Wir können die s-te Spalte als Pivotzeile wählen, falls $\bar{c}_s < 0$.
- Eine übliche Wahl ist das minimale $\bar{c}_s$ (steepest descent).
- Falls $\bar{a}_{is} \leq 0$ für alle $i \in \{1, \ldots, m\}$, dann ist die minimale Lösung unbeschränkt.
- Andernfalls wählt man t als Pivotspalte, so daß $\bar{b}_t / \bar{a}_{ts}$ minimal für $\bar{a}_{ts} > 0$ ist.
- Falls es mehrere gleich gute $\bar{b}_t / \bar{a}_{ts}$ gibt, dann gelangt man zu einer entarteten Basis. Durch eine *Pivotstrategie* muß eines der t gewählt werden.

Pivotstrategien

Eine übliche Strategie ist es, nachdem s durch steepest-descent festgelegt wurde, unter den möglichen t das erste zu nehmen.

Dabei kann es aber zu einem Zyklus kommen, der aus einer entarteten Ecke nicht mehr herauskommt. Dies passiert aber in der Praxis so gut wie nie.

Durch zufällige Wahl eines Kandidaten läßt sich dieses pathologische Verhalten einfach verhindern.

Es gibt deterministische Strategien, die einen Zyklus ausschließen. Sie sind aber umständlich zu implementieren und scheinen langsamer zur optimalen Lösung zu gelangen.

Der Zweiphasenalgorithmus

Es sei ein Problem in der Form $Ax \leq b, x \geq 0$ mit $b \geq 0$ gegeben.

Dann ist das zugehörige Problem in Standardform
$Ax + Iy = b, x \geq 0, y \geq 0$.

Dafür ist $y = b, x = 0$ eine zulässige Basislösung, falls wir y als Basisvariablenvektor wählen. Wir können sofort mit dem Simplexalgorithmus starten.

Was machen wir aber, wenn wir ein allgemeines Problem lösen wollen, für das eine zulässige Basislösung nicht bekannt ist?

Der Zweiphasenalgorithmus

Es sei $c^T x$ zu minimieren unter $Ax = b, x \geq 0$ (Problem A).

Konstruiere ein neues Problem (Problem B):

Minimiere $s_1 + \cdots + s_m$ unter $Ax + Is = b, x \geq 0, y \geq 0$.

Theorem

Es gibt eine zulässige Basislösung zu Problem A genau dann, wenn $s = 0$ die optimale Lösung von Problem B ist. Ist (x, s) die Lösung von B, dann ist x eine zulässige Basislösung von A.

Der Zweiphasenalgorithmus

Es sei $c^T x$ zu minimieren unter $Ax = b, x \geq 0$.

1. Negiere alle Zeilen i mit $b_i < 0$.
2. Minimiere $s_1 + \cdots + s_m$ unter $Ax + Is = b, x \geq 0, y \geq 0$ mit dem Simplexalgorithmus. Nehme $x = 0, s = b$ als zulässige Anfangsbasislösung.
3. Falls $s_1 + \cdots + s_m > 0$, dann gibt es keine Lösung zum Ursprungsproblem.
4. Sonst ist jetzt ein $x \geq 0$ bekannt, mit $Ax = b$.
5. Aus diesem x ergibt sich eine zulässige Basislösung.
6. In der zweiten Phase kann jetzt das ursprüngliche Problem mit dem Simplexalgorithmus gelöst werden.

Dualität

Dualität beschäftigt sich mit Paaren von linearen Programmen und der Beziehung zwischen ihnen.

Das *primale Problem* sei

$$z = \max\{\, c^T x \mid Ax \le b,\ x \ge 0 \,\}. \tag{P}$$

Das zugehörige *duale Problem* ist dann definiert als

$$w = \min\{\, b^T u \mid A^T u \ge c,\ u \ge 0 \,\} \tag{D}$$

oder, anders geschrieben,

$$w = \min\{\, u^T b \mid u^T A \ge c^T,\ u \ge 0 \,\}.$$

Beispiel

$$\begin{array}{lrcrcrcr}
\text{Max} & 7x_1 & + & 2x_2 & & & & \\
 & -\ x_1 & + & 2x_2 & & & \le & 4 \\
 & 5x_1 & + & x_2 & & & \le & 20 \\
 & -\ 2x_1 & - & 2x_2 & & & \le & -7 \\
 & & & x_1, x_2 & & & \ge & 0
\end{array}$$

$$\begin{array}{lrcrcrcr}
\text{Min} & 4u_1 & + & 20u_2 & - & 7u_3 & & \\
 & -\ u_1 & + & 5u_2 & - & 2u_3 & \ge & 7 \\
 & 2u_1 & + & u_2 & - & 2u_3 & \ge & 2 \\
 & & & & & u_1, u_2, u_3 & \ge & 0
\end{array}$$

Dualität

Theorem

Das duale Problem des dualen Problems ist wieder das primale.

Beweis Wir beginnen mit dem dualen Problem:

$$\begin{aligned} w &= \min\{\, u^T b \mid u^T A \ge c^T,\ u \ge 0 \,\} \\ &= \max\{\, -b^T x \mid -A^T x \le -c,\ x \ge 0 \,\} \end{aligned}$$

Das duale Problem hiervon ist:

$$\begin{aligned} &= \min\{\, -c^T x \mid -Au \ge -b,\ u \ge 0 \,\} \\ &= \max\{\, c^T x \mid Ax \le b,\ x \ge 0 \,\} \end{aligned}$$

Das ist jetzt das primale Problem. □

Schwache Dualität

Sei wieder

$$z = \max\{\, c^T x \mid Ax \le b,\ x \ge 0 \,\} \tag{P}$$

und

$$w = \min\{\, b^T u \mid A^T u \ge c,\ u \ge 0 \,\}. \tag{D}$$

x ist *primal zulässig*, wenn $x \in \{\, x \mid Ax \le b, x \ge 0 \,\}$.

u ist *dual zulässig*, wenn $u \in \{\, u \mid u^T A \ge c^T, u \ge 0 \,\}$.

Theorem (Schwache Dualität)

Sei $\hat{x}$ primal zulässig und $\hat{u}$ dual zulässig. Dann gilt

$$c^T \hat{x} \le z \le w \le b^T \hat{u}.$$

Schwache Dualität

Beweis

$$c^T\hat{x} \leq \hat{u}^T A\hat{x} \leq b^T\hat{u}, \text{ denn}$$

1. $\hat{u}^T A \geq c^T$ und $\hat{x} \geq 0$
2. $A\hat{x} \leq b$ und $\hat{u} \geq 0$

Da $z = c^T x$ für ein primal zulässiges x und $w = b^T u$ für ein dual zulässiges u, folgt $z \leq w$.

Korollar

Falls (P) eine unbeschränkte optimale Lösung hat, dann hat (D) keine Lösung.

Dualität

Folgende Paare sind dual:

$$z = \min\{\, c^T x \mid Ax = b,\ x \geq 0\,\} \qquad (P)$$

$$w = \max\{\, b^T y \mid A^T y \leq c\,\} \qquad (D)$$

Beweis:

$$\begin{aligned} z &= \min\{\, c^T x \mid Ax = b,\ x \geq 0\,\} \\ &= \min\left\{ c^T x \,\middle|\, \begin{pmatrix} A \\ -A \end{pmatrix} x \leq \begin{pmatrix} b \\ -b \end{pmatrix},\ x \geq 0 \right\} \end{aligned}$$

$$\begin{aligned} z &= \min\{\, c^T x \mid Ax = b,\ x \geq 0\,\} \\ &= \max\left\{ -c^T x \,\middle|\, \begin{pmatrix} A \\ -A \end{pmatrix} x \leq \begin{pmatrix} b \\ -b \end{pmatrix},\ x \geq 0 \right\} \end{aligned}$$

Das duale Problem ist nach Definition:

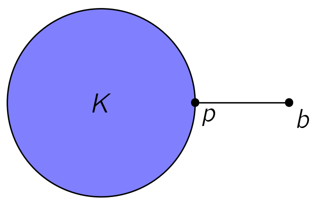
$$\begin{aligned} w &= \min\left\{ (b^T, -b^T)\begin{pmatrix} y \\ \bar{y} \end{pmatrix} \,\middle|\, (A^T, -A^T)\begin{pmatrix} y \\ \bar{y} \end{pmatrix} \geq -c,\ y, \bar{y} \geq 0 \right\} \\ &= \min\{\, b^T y - b^T \bar{y} \mid A^T y \geq -c,\ -A^T \bar{y} \geq -c,\ y, \bar{y} \geq 0\,\} \\ &= \max\{\, b^T y \mid A^T y \leq c,\ y \leq 0\,\} + \max\{\, b^T \bar{y} \mid A^T \bar{y} \leq c,\ \bar{y} \geq 0\,\} \\ &= \max\{\, b^T y \mid A^T y \leq c\,\} \end{aligned}$$

$\square$

Lemma (Das Projektionstheorem)

Sei K eine abgeschlossene, konvexe Teilmenge des $\mathbf{R}^n$ und $b \in \mathbf{R}^n$.

Sei p die *Projektion* von b auf K (d.h. der Punkt in K mit minimalem Euklidischen Abstand $|p - b|$ von b).

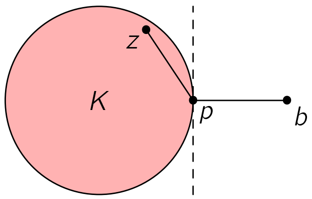


Dann gilt

$$(z - p)^T (b - p) \leq 0$$

für alle $z \in K$.

Beweis



Da K konvex ist, liegen z und b auf verschiedenen Seiten der Hyperebene, die normal zu $b - p$ durch p geht (wobei z auch auf dieser Hyperebene liegen kann).

Sei α der Winkel zwischen $z - p$ und $b - p$. Dann ist $\alpha \geq \pi/2$.

$\Rightarrow (z-p)^T(b-p) = |z-p| \cdot |b-p| \cos\alpha \leq 0$.

(Falls $b \in K$, dann gilt die Aussage trivialerweise.)

Dualität

Theorem (Farkas Lemma A)

Sei $A \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$. Dann ist genau eine dieser Aussagen wahr:

1. Es gibt ein $x \in \mathbf{R}^n$ mit $Ax = b$, $x \geq 0$.
2. Es gibt ein $y \in \mathbf{R}^m$ mit $A^T y \geq 0$, $b^T y < 0$.

Beweis

Angenommen beide Aussagen gälten.

$$Ax = b \Rightarrow y^T Ax = y^T b \Rightarrow x^T A^T y = y^T b < 0 \tag{1}$$

$$x \geq 0,\ A^T y \geq 0 \Rightarrow x^T A^T y \geq 0 \tag{2}$$

Offensichtlich widersprechen sich (1) und (2). Also kann höchstens eine der Aussagen wahr sein.

Nehmen wir jetzt an, die erste Aussage gilt nicht.

Sei $K = \{\, Ax \mid x \geq 0 \,\}$. K ist abgeschlossen und konvex und $b \notin K$.

Sei p die Projektion von b auf K.

Es gibt ein w mit $Aw = p$.

Gemäß Projektionstheorem gilt dann

$$(z-p)^T(b-p) \leq 0 \text{ für alle } z \in K.$$

Insbesondere gilt

$$(Ax-p)^T(b-p) \leq 0 \text{ für alle } x \geq 0.$$

Wir wählen $y = p - b$. Dann gilt

$$\begin{aligned}
(Ax-p)^T(b-p) &\leq 0 \text{ für alle } x \geq 0 \\
\Rightarrow (Ax-p)^T y &\geq 0 \text{ für alle } x \geq 0 \\
\Rightarrow (Ax-Aw)^T y &\geq 0 \text{ für alle } x \geq 0 \\
\Rightarrow (x-w)^T A^T y &\geq 0 \text{ für alle } x \geq 0 \\
\Rightarrow x^T A^T y &\geq 0 \text{ für alle } x \geq 0 \text{ (denn } w \geq 0) \\
\Rightarrow A^T y &\geq 0
\end{aligned}$$

Damit haben wir $A^T y \geq 0$ bewiesen.

Wir haben $A^T y \geq 0$ und brauchen noch $b^T y < 0$.

$$y^T b = (p - y)^T y = p^T y - y^T y$$

Es gilt aber

$$(Ax - p)^T y \geq 0$$

für alle $x \geq 0$ (vorige Folie).

Insbesondere ist $p^T y \leq 0$ (aus $x = 0$).

Es ist $y^T y > 0$, denn $y = p - b \neq 0$.

$$\Rightarrow y^T b = p^T y - y^T y < 0$$

□

Dualität

Theorem (Farkas Lemma B)

Sei $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$. Dann ist genau eine dieser Aussagen wahr:

1. Es gibt ein $x \in \mathbf{R}^n$ mit $Ax \leq b$.
2. Es gibt ein $y \geq 0$ mit $A^T y = 0$, $b^T y < 0$.

Beweis

Übungsaufgabe!

Starke Dualität

$$z = \min\{\, c^T x \mid Ax = b,\ x \geq 0 \,\} \qquad (P)$$

$$w = \max\{\, b^T y \mid A^T y \leq c \,\} \qquad (D)$$

Theorem (Starke Dualität)

Falls das primale oder duale Problem zulässige Lösungen besitzt, dann gilt $z = w$.

Beweis

Wir müssen lediglich $z \leq w$ zeigen.

O.B.d.A. habe P eine zulässige Lösung (Dualität!).

Falls P unbeschränkt ist, dann $z = w = -\infty$ wegen schwacher Dualität.

Beweis (Fortsetzung)

Sei nun P beschränkt und x^* eine optimale primale Lösung. Es gilt daher $Ax^* = b$ und $c^T x^* = z$.

Wenn wir zeigen können, daß es ein y gibt, mit

$$A^T y \leq c \text{ und } b^T y \geq z,$$

dann sind wir fertig, denn y ist eine zulässige duale Lösung. Der Wert der Zielfunktion ist mindestens z. Also ist $w \geq z$.

Nehmen wir also an, so ein y existiere nicht.

Wir verwenden jetzt diese Form von Farkas Lemma:

1. Entweder $\exists \hat{x}$ mit $\hat{A}\hat{x} \leq \hat{b}$,
2. oder $\exists \hat{y} \geq 0$ mit $\hat{A}^T\hat{y} = 0$ und $\hat{b}^T\hat{y} < 0$.

Wir definieren:

$$\hat{A} := \begin{pmatrix} A^T \\ -b^T \end{pmatrix}, \quad \hat{b} := \begin{pmatrix} c \\ -z \end{pmatrix}, \quad \hat{x} := y, \quad \begin{pmatrix} x \\ \lambda \end{pmatrix} := \hat{y}$$

Nach unserer Annahme ist dann 1. im Lemma falsch und 2. muß wahr sein.

Die Annahme war: Es gibt kein y mit $A^Ty \leq c$ und $b^Ty \geq z$.

Es gilt also 2.: $\exists \hat{y} \geq 0$ mit $\hat{A}^T\hat{y} = 0$ und $\hat{b}^T\hat{y} < 0$, wobei

$$\hat{A} := \begin{pmatrix} A^T \\ -b^T \end{pmatrix}, \quad \hat{b} := \begin{pmatrix} c \\ -z \end{pmatrix}, \quad \hat{x} := y, \quad \begin{pmatrix} x \\ \lambda \end{pmatrix} := \hat{y}.$$

Das bedeutet aber gerade, daß es $x \geq 0$ und $\lambda \geq 0$ gibt mit

$$Ax = \lambda b \text{ und } c^Tx < \lambda z.$$

Falls $\lambda > 0$, dann gilt $Ax' = b$ und $c^Tx' < z$ mit $x' = \frac{1}{\lambda}x$. Das ist ein Widerspruch zur Minimalität von z.

Falls aber $\lambda = 0$, dann ist $A(x^* + x) = b$ und $c^T(x^* + x) = z + c^Tx < z$. Das ist auch ein Widerspruch zur Minimalität von z. $\square$

Die Größe eines Linearen Programms

Sei $k \in \mathbf{Z}$, $x \in Z^m$, $M \in \mathbf{Z}^{m \times n}$.

Wir definieren die *Größe einer Kodierung* als

$$\begin{aligned} size(k) &:= 1 + \lceil \log_2(|k| + 1) \rceil \\ size(x) &:= \sum_{i=1}^{m} size(x_i) \\ size(M) &:= \sum_{i=1}^{m} \sum_{j=1}^{n} size(M_{ij}) \end{aligned}$$

Die Größe eines Linearen Programms

Sei $A \in \mathbf{Z}^{m \times n}$, $b \in \mathbf{Z}^m$, $c \in \mathbf{Z}^n$.

$$\begin{aligned} \text{Minimiere } & c^Tx \\ \text{unter } & Ax = b \\ & x \geq 0 \end{aligned} \qquad (LP)$$

Größe dieses linearen Programms:

$$size(LP) := size(A) + size(b) + size(c)$$

(Durch Skalieren mit dem lcp läßt sich jedes LP in diese Form bringen.)

Die Größe eines Linearen Programms

Statt mit $size(LP)$ rechnet es sich oft günstiger mit einer Größe L, die so definiert ist:

$$L := size(det_{\max}) + size(b_{\max}) + size(c_{\max}) + m + n,$$

wobei

$$det_{\max} = \max_{A'} |det A'| \quad (A' \text{ Untermatrix von } A)$$
$$b_{\max} = \max_i |b_i|$$
$$c_{\max} = \max_i |c_i|$$

Behauptung:

$$L < size(LP)$$

$\Rightarrow$ Ist die Laufzeit polynomiell in L, dann auch in $size(LP)$.

Größe der Ausgabe

Sei x eine Ecke von $\{\, x \mid Ax = b,\, x \geq 0 \,\}$.

Dann ist

$$x^T = \begin{pmatrix} \frac{p_1}{q} & \frac{p_2}{q} & \cdots & \frac{p_n}{q} \end{pmatrix},$$

mit

$$p_i, q \in \mathbf{N},\ 0 \leq p_i < 2^L,\ 1 \leq q < 2^L.$$

Die Ausgabe ist also polynomiell repräsentierbar.

Interior Point Methods

Theorem

Seien x_1 und x_2 Ecken von $\{\, x \mid Ax = b,\, x \geq 0 \,\}$.

Falls $c^T x_1 \neq c^T x_2$, dann ist $|c^T x_1 = c^T x_2| > 2^{-2L}$.

Korollar

Sei $z = \min\{\, c^T x \mid Ax = b,\, x \geq 0 \,\}$ und sei x eine zulässige Lösung mit $c^T x \leq z + 2^{-2L}$.

Dann ist jede Ecke x' mit $c^T x' \leq c^T x$ eine optimale Lösung.

$\Rightarrow$ Es genügt nahe an eine Lösung heranzukommen und man muß nur mit einer beschränkten Genauigkeit rechnen.

Interior Point Methods

Wir lösen gleichzeitig dieses primale und sein duales Problem:

$$\begin{aligned} \text{Minimiere } z &= c^T x \\ \text{unter } Ax &= b \\ x &\geq 0 \end{aligned} \qquad (P)$$

$$\begin{aligned} \text{Maximiere } w &= b^T y \\ \text{unter } A^T y + s &= c \\ s &\geq 0 \end{aligned} \qquad (D)$$

Der Algorithmus arbeitet mit einer primalen Lösung $\bar{x} > 0$ und einer Belegung $\bar{s} > 0$ von s, zu welcher es ein $\bar{y}$ mit $A^T \bar{y} + \bar{s} = c$ gibt.

Interior Point Methods

Eine Grundidee des Algorithmus ist es im Inneren des Polyeders zu bleiben. Keine Komponente von $\bar{x} > 0$ und $\bar{s} > 0$ soll also klein werden.

Folgende Abbildung transformiert das Problem in einen Bildraum:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \mapsto x' = \begin{pmatrix} x_1/\bar{x}_1 \\ x_2/\bar{x}_n \\ \vdots \\ x_n/\bar{x}_n \end{pmatrix}$$

Diese Abbildung bildet $\bar{x}$ auf e (den Einsvektor) ab.

Skalierung

Wir können diese Abbildung auch so beschreiben:

$$x \mapsto x' = \bar{X}^{-1}x$$

mit

$$\bar{X} = \begin{pmatrix} \bar{x}_1 & 0 & 0 & \cdots & 0 \\ 0 & \bar{x}_2 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & \bar{x}_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & \bar{x}_{n-1} \end{pmatrix}.$$

Skalierung

Im Bildraum sehen die Probleme so aus:

$$\begin{aligned} \text{Minimiere } z &= \bar{c}^T x' \\ \text{unter } \bar{A}x' &= b \\ x' &\geq 0 \end{aligned} \qquad (P)$$

$$\begin{aligned} \text{Maximiere } w &= b^T y \\ \text{unter } \bar{A}^T y + s' &= \bar{c} \\ s' &\geq 0 \end{aligned} \qquad (D)$$

Dabei ist $\bar{A} = A\bar{X}$ und $\bar{c} = \bar{X}c$.

Die Dualitätslücke

Es stellt sich heraus, daß

$$s' = \bar{X}s = \begin{pmatrix} s_1\bar{x}_1 \\ s_2\bar{x}_2 \\ \vdots \\ s_n\bar{x}_n \end{pmatrix}.$$

Die *Dualitätslücke* $c^T\bar{x} - b^T\bar{y} = \bar{x}^T\bar{s}$ ändert sich durch Skalierung nicht: Es gilt ja $x^Ts = x'^Ts'$.

(Sobald die Dualitätslücke sehr klein ist, kann der Algorithmus terminieren.)

Die Potentialfunktion

Definition

$$G(x,s) = (n+\sqrt{n})\ln(x^T s) - \sum_{j=1}^{n} \ln(x_j s_j)$$

1. Die Potentialfunktion ist unter Skalierung invariant
2. Man kann leicht einen zulässigen Startwert $\bar{x} > 0$, $\bar{s} > 0$ finden, mit $G(\bar{x},\bar{s}) = O(\sqrt{n}L)$
3. Ist $G(\bar{x},\bar{s}) \leq -2\sqrt{n}L$ dann sind wir nah am Optimum und können stoppen.
4. Wir können stets einen *Schritt* $\bar{x},\bar{s} \mapsto \tilde{x},\tilde{s}$ machen, so daß $G(\tilde{x},\tilde{s}) - G(\bar{x},\bar{s}) \leq -7/120$.

$\Rightarrow$ Nach $O(\sqrt{n}L)$ Iterationen sind wir fertig.

Der primale Schritt

Wir machen entweder einen *primalen* oder einen *dualen Schritt*.

Der primale Schritt ändert nur $\bar{x}$ und sieht so aus:

Zunächst gehen wir in den Bildraum (unser Punkt ist jetzt (e, s')).

Jetzt möchten wir in eine Richtung gehen, in der die Potentialfunktion um möglichst viel kleiner wird.

Dazu betrachten wir den Gradienten von $G(x,s)$ am Punkt (e,s'):

$$\begin{aligned} g &= \nabla_x G(x,s)|_{(e,s')} \\ &= \left.\frac{n+\sqrt{n}}{x^T s} s - \begin{pmatrix} 1/x_1 \\ \vdots \\ 1/x_n \end{pmatrix}\right|_{(e,s')} \\ &= \frac{n+\sqrt{n}}{e^T s'} s' - e \end{aligned}$$

Wir können aber nicht in Richtung $-g$ gehen, da wir den zulässigen Bereich verlassen könnten.

Sei d die Projektion von g auf $\{\, x \mid \bar{A}x = 0 \,\}$.

Wir können in Richtung $-d$ gehen!

Der primale Schritt

Der Vektor d läßt sich so ausdrücken:

$$d = (I - \bar{A}(\bar{A}\bar{A}^T)^{-1}\bar{A})g$$

Der primale Schritt geht zu

$$\tilde{x} = e - \frac{1}{4|d|}d, \;\; \tilde{s} = s'.$$

(Danach muß wieder zurückskaliert werden.)

Um zu garantieren, daß die Potentialfunktion genügend abnimmt (um 7/120), muß $|d| = \sqrt{d^T d} \geq 0.4$ gelten. Nur wenn dies der Fall ist, machen wir einen primalen Schritt.

Der duale Schritt

Falls $|d| < 0.4$ ist, machen wir einen *dualen Schritt*, der nur s' verändert. Der Gradient ist jetzt:

$$h = \nabla_s G(x,s)|_{(e,s')} = \frac{n+\sqrt{n}}{e^T s'} e - \begin{pmatrix} 1/s'_1 \\ \vdots \\ 1/s'_n \end{pmatrix}$$

Wir brauchen: Es muß ein y mit $\bar{A}^T y + \tilde{s} = \bar{c}$ geben.

Es geht, wenn wir den Schritt so wählen:

$$\tilde{s} = \frac{e^T s'}{n+\sqrt{n}}(d+e), \;\; \tilde{x} = x' = e$$

Man kann zeigen, daß $\tilde{s} > 0$ und daß die Potentialfunktion um mindestens $1/6$ abnimmt.

Ganzzahliges Programmieren (ILP)

Sei A eine rationale Matrix und b, c rationale Vektoren. Das *ILP* ist es, folgende Optimierungsaufgaben zu lösen:

$$\begin{aligned} \text{Maximiere } & c^T x \\ \text{unter } & Ax \le b \\ & x \text{ ganzzahlig} \end{aligned}$$

oder

$$\begin{aligned} \text{Maximiere } & c^T x \\ \text{unter } & Ax = b \\ & x \ge 0 \\ & x \text{ ganzzahlig} \end{aligned}$$

Eine einfache Abschätzung

Dualität liefert:

$$\begin{aligned} & \max\{\, c^T x \mid Ax \le b,\; x \text{ ganzzahlig}\,\} \\ \le & \min\{\, b^t y \mid A^T y = c,\; y \ge 0,\; y \text{ ganzzahlig}\,\} \end{aligned}$$

Im Gegensatz zu linearem Programmieren, ist die Ungleichung bei ILP normalerweise echt. Durch die *LP-Relaxation* erhalten wir eine Abschätzung, falls die Vorraussetzung für starke Dualität vorliegt:

$$\begin{aligned} & \max\{\, c^T x \mid Ax \le b,\; x \text{ ganzzahlig}\,\} \\ \le & \max\{\, c^T x \mid Ax \le b\,\} = \min\{\, b^t y \mid A^T y = c,\; y \ge 0\,\} \\ \le & \min\{\, b^t y \mid A^T y = c,\; y \ge 0,\; y \text{ ganzzahlig}\,\} \end{aligned}$$

Beispiel

Sei $A = (2)$, $b = (1)$, $c = (1)$.

$$\begin{aligned} & \max\{\, c^T x \mid Ax \le b,\; x \text{ ganzzahlig}\,\} && = 0 \\ \le & \max\{\, c^T x \mid Ax \le b\,\} && = 1/2 \\ = & \min\{\, b^t y \mid A^T y = c,\; y \ge 0\,\} && = 1/2 \\ \le & \min\{\, b^t y \mid A^T y = c,\; y \ge 0,\; y \text{ ganzzahlig}\,\} && = \text{exisiert nicht} \end{aligned}$$

Beispiel

Sei $A = (2)$, $b = (1)$, $c = (1)$.

$$
\begin{aligned}
& \max\{\, c^T x \mid Ax \leq b,\ x \geq 0\, x \text{ ganzzahlig} \,\} && = 0 \\
\leq\, & \max\{\, c^T x \mid Ax \leq b,\ x \geq 0 \,\} && = 1/2 \\
=\, & \min\{\, b^t y \mid A^T y \geq c,\ y \geq 0 \,\} && = 1/2 \\
\leq\, & \min\{\, b^t y \mid A^T y \geq c,\ y \geq 0,\ y \text{ ganzzahlig} \,\} && = 1
\end{aligned}
$$

Komplexität von ILP

Theorem

ILP ist *NP*-vollständig

Beweis

Reduktion von 3SAT auf ILP.

Gegeben ist eine Instanz F von 3SAT bestehend aus n Variablen $x_1, \ldots x_n$ und m Klauseln der Form $\{l_1, l_2, l_3\}$, wobei $l_i \in \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$.

Wir konstruieren aus F ein ILP I mit:

F hat eine erfüllende Belegung $\iff$ der optimale Wert von I ist 0

Unser ILP I hat die Literale von F als Variablen und sieht so aus:

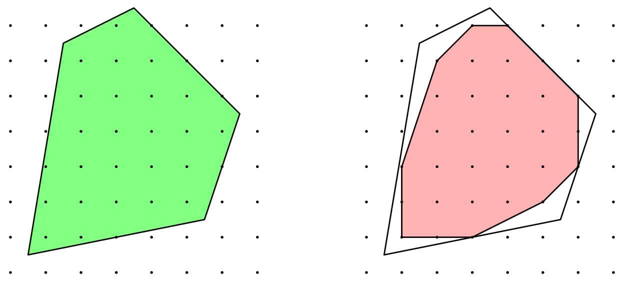
Maximiere 0

unter folgenden Bedingungen:

$$
\begin{aligned}
l_1 + l_2 + l_3 \geq 1 & \text{ für alle Klauseln } \{l_1, l_2, l_3\} \text{ in } F \\
0 \leq x_i, \bar{x}_i \leq 1 & \text{ für } i = 1, \ldots, n \\
x_i + \bar{x}_i = 1 & \text{ für } i = 1, \ldots, n \\
x_1, \ldots, x_n & \text{ ganzzahlig}
\end{aligned}
$$

Es gibt genau dann eine Lösung, wenn F erfüllbar ist. Die Konstruktion ist in polynomieller Zeit durchführbar. □

Die ganzzahlige Hülle



Die *ganzzahlige Hülle* P_I eines Polyeders P ist die konvexe Hülle aller ganzzahliger Punkte in P.

Falls alle Ecken von P ganzzahlig sind, das ist $P = P_I$ und das ILP kann durch lineares Programmieren gelöst werden.

Ein einfaches Lösungsverfahren

Wir wollen $\max\{\, c^T x \mid Ax \leq b,\ x \text{ ganzzahlig} \,\}$ bestimmen.

Wir starten mit $\Pi_1 = P$, $P = \{\, x \mid Ax \leq b \,\}$.

Im Schritt k haben wir eine Menge $\Pi_k = \{P_1, \dots, P_k\}$ von Polyedern mit

1. $P_1, \dots, P_k$ sind paarweise disjunkt (repräsentiert durch lineare Ungleichungen),
2. jeder ganzzahlige Punkt in P ist in $P_1 \cup \dots \cup P_k$ enthalten.

Ein Schritt sieht so aus (Eingabe Π_k):

Sei $\mu_j = \max\{\, c^T x \mid x \in P_j \,\}$ und j^* so, daß μ_{j^*} maximal unter allen μ_j ist.

Sei x^* so, daß $\mu_{j^*} = c^T x^*$ (diese Werte lassen sich durch lineares Programmieren finden).

Falls x^* ganzzahlig ist, dann ist x^* eine optimale Lösung.

Andernfalls, sei x_i^* eine nicht-ganzzahlige Komponente. Wir definieren

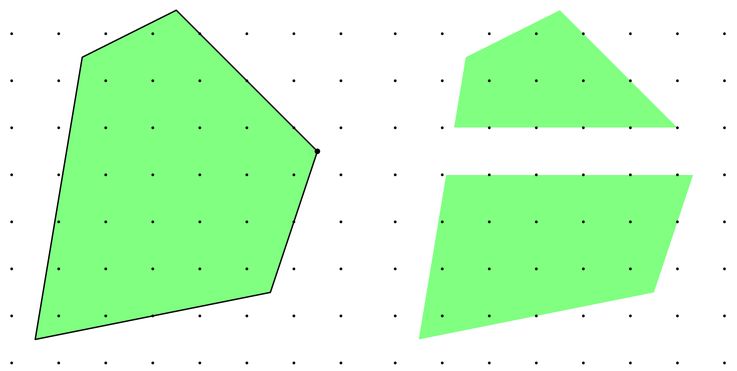
$$Q_1 = \{\, x \mid P_{j^*} \mid x_i \geq \lceil x_i^* \rceil \,\}$$
$$Q_2 = \{\, x \mid P_{j^*} \mid x_i \leq \lfloor x_i^* \rfloor \,\}$$

Wir setzen

$$\Pi_{k+1} := \{P_1, \dots, P_{j^*-1}, Q_1, Q_2, P_{j^*+1}, \dots, P_k\}.$$

(Falls $P_i = \emptyset$ für alle i, dann gibt es keine Lösung.)

Beispiel



NP-vollständige Probleme

- Keine polynomiellen Algorithmen, falls $P \neq NP$.
- Viele wichtige Probleme sind *NP*-vollständig.
- Irgendwie müssen sie gelöst werden.
- Es reicht nicht zu sagen, daß das Problem *NP*-hart ist und daher nicht gelöst werden kann!

Approximationsalgorithmen

In polynomieller Zeit lässen sich nicht exakte Lösungen von NP-harten Problemen berechnen.

Approximationsalgorithmen versuchen das beste aus einer Polynomialzeitberechnung zu machen, indem sie wenigstens eine Näherunslösung bestimmen.

Im Gegensatz zu Heuristiken, verlangen wir aber von Approximationsalgorithmen eine Garantie, daß die berechnete Näherungslösung nah am Optimum ist.

Approximationsalgorithmen

Sei A ein Algorithmus, der in polynomieller Zeit eine Näherungslösung zu einem gegebenen Optimierungsproblem berechnet.

Sei $F^*(I)$ der optimale Wert der Zielfunktion und $F(I)$ der vom Algorithmus A erzielte Wert.

Definition

Das *Approximationsverhältnis* von A ist α, wenn

$$\frac{F(I) - F^*(I)}{F^*(I)} \leq \alpha \text{ und } \frac{F^*(I) - F(I)}{F^*(I)} \leq \alpha$$

gilt.

Approximationsalgorithmen

Diese Definition ist sowohl für Minimierungs- als auch für Maximierungsprobleme sinnvoll.

Für Minimierungsprobleme ist diese Ungleichung ausschlaggebend:

$$\frac{F(I) - F^*(I)}{F^*(I)} \leq \alpha$$

Für Maximierungsprobleme dagegen die andere:

$$\frac{F^*(I) - F(I)}{F^*(I)} \leq \alpha$$

Beispiel: Vertex Cover

Das Approximationsverhältnis ist 2.

```
C := ∅;
while E ≠ ∅ do
  choose some e ∈ E;
  V := V − e;
  C := C ∪ e;
  E := { e′ ∈ E | e ∩ e′ = ∅ }
od;
return C
```

Kompetitive Analyse

Um das Approximationsverhältnis zu analysieren, bietet sich eine *kompetitive Analyse* der folgenden Form an:

Wir gehen von einer Instanz und einer zugehörigen optimalen Lösung aus.

Wir müssen dann beweisen, daß der Approximationsalgorithmus eine Lösung findet, die nur um α schlechter als die vorgegebene optimale Lösung ist.

Kompetitive Analyse

Im Falle von Vertex Cover ist eine kompetitive Analyse recht einfach:

Sei $G = (V, E)$ ein Graph und $C \subseteq V$ ein minimales Vertex Cover.

Der Algorithmus wählt eine Kante $e = \{v_1, v_2\} \in E$ aus fügt v_1 und v_2 zu seinem VC hinzu. Einer von beiden Knoten ist aber auch in C. Zu einem Knoten in C, fügt der Algorithmus also höchstens 2 zu seiner Lösung hinzu.

Das macht ein Verhältnis von höchstens 2 zwischen konstruierter und optimaler Lösung aus.

Approximationsalgorithmen

Definition

1. Ein Approximationsalgorithmus ist ein *$f(n)$-Approximationsalgorithmus*, wenn das Approximationsverhältnis höchstens $f(n)$ für alle Instanzen der Länge n ist.
2. Ein *ϵ-Approximationsalgorithmus* ist ein $f(n)$-Approximationsalgorithmus mit $f(n) \leq \epsilon$ für alle $n \in \mathbf{N}$.

Approximationsschemata

Definition

Sei $A(\epsilon)$ ein Approximationsalgorithmus mit einem Parameter ϵ.

1. $A(\epsilon)$ ist ein *PTAS (polynomial time approximation scheme)*, falls $A(\epsilon)$ ein ϵ-Approxmiationsalgorithmus für jedes $\epsilon > 0$ ist und polynomielle Laufzeit in der Eingabelänge für jedes $\epsilon > 0$ hat.
2. $A(\epsilon)$ ist ein *FPTAS (fully polynomial time approximation scheme)*, falls $A(\epsilon)$ ein ϵ-Approxmiationsalgorithmus für jedes $\epsilon > 0$ ist und polynomielle Laufzeit in der Eingabelänge und $1/\epsilon$ für jedes $\epsilon > 0$ hat.

Scheduling

Wir betrachten folgendes Problem:

Gegeben sind n Aufgaben und m identische Prozessoren, sowie die Ausführungszeiten $t_1, \ldots, t_n$ der n Aufgaben.

Gesucht ist eine Verteilung der Aufgaben auf die m Prozessoren, so daß die *Schlußzeit* minimiert wird, also die Zeit, bis alle Aufgaben erledigt wurden.

- Wir nennen dieses Problem im folgenden *Scheduling*.
- Für $m = 1$ ist es sehr leicht zu lösen.
- Ab $m = 2$ ist es NP-vollständig.

Beispiel

Sei $m = 3$, $n = 7$ und $(t_1, \ldots, t_6) = (8, 7, 6, 5, 4, 3)$.

Dieser Schedule ist optimal:

P_1	8	3
P_2	7	4
P_3	6	5

Die Optimalität folgt hier aus der Tatsache, daß alle Prozessoren gleichzeitig fertig werden.

Die LPT-Regel

Ein sehr einfaches Verfahren, in polynomieller Zeit einen Schedule zu erzeugen, nennt sich *LPT = longest processing time*.

Definition

Die LPT-Regel weist dem nächsten freiwerdenden Prozessor die Aufgabe mit der größten Bearbeitungszeit zu. (Gibt es mehrere gleich große, dann nehmen wir irgendeine von ihnen.)

Beispiel

Sei $m = 3$, $n = 7$, $(t_1, \ldots, t_7) = (5, 5, 4, 4, 3, 3, 3)$.

Ein optimaler Plan ist offensichtlich:

5	4	
5	4	
3	3	3

Die LPT-Regel liefert dagegen:

5	3	3
5	3	
4	4	

Natürlich kann die LPT-Regel nicht immer optimale Pläne konstruieren.

Wie gut ist aber ein LPT-Schedule?

LPT-Schedules

Theorem

Sei I eine Instanz des m-Prozessorscheduling und $F^*(I)$ die optimale Schlußzeit. Sei $F(I)$ die Schlußzeit eines LPT-Schedules.

Dann gilt:

$$\frac{F(I) - F^*(I)}{F^*(I)} \leq \frac{1}{3} - \frac{1}{3m}$$

Insbesondere liefert die LPT-Regel einen 1/3-Approximationsalgorithmus.

Beweis

Für $m = 1$ ist ein LPT-Schedule optimal, also $F(I) = F^*(I)$. Sei nun $m > 1$.

Nehmen wir an, daß Theorem ist falsch. Dann gibt es eine Instanz $(t_1, \ldots, t_n)$, für die der LPT-Schedule schlechter ist, als vom Theorem behauptet.

Wir können annehmen, daß das Theorem für alle Instanzen mit weniger als n Aufgaben gilt und daß $t_1, \ldots, t_n$. Die LPT-Regel soll die Aufgaben in der Reihenfolge ihrer Indizes zuordnen.

Sei $F(I)$ die Schlußzeit des LPT-Schedules.

Behauptung 1: Die Aufgabe mit Index n wird genau zur Schlußzeit, aber nicht früher beendet.

Wenn nicht, dann gibt es eine Aufgabe $k < n$, die später als Aufgabe n zur Schlußzeit beendet wird. Sei I' die Instanz, die aus den Aufgaben $1, \ldots, k$ besteht.

$$\frac{F(I') - F^*(I')}{F^*(I')} \geq \frac{F(I) - F^*(I)}{F^*(I)} > \frac{1}{3} - \frac{1}{3m}$$

Dies ist ein Widerspruch dazu, daß das Theorem für weniger als n Aufgaben gilt.

Behauptung 2: In einem optimalen Plan für I werden nicht mehr als zwei Aufgaben von einem Prozessor erledigt.

Wegen Behauptung 1 wird die Aufgabe n zum Zeitpunkt $F(I) - t_n$ im LPT-Schedule gestartet. Bis zu diesem Zeitpunkt müssen alle Prozessoren beschäftigt sein, denn sonst wäre Aufgabe n früher zugeordnet worden. Daraus folgt:

$$F(I) - t_n \leq \frac{1}{m} \sum_{n=1}^{n-1} t_i$$

$$F(I) \leq \frac{1}{m} \sum_{n=1}^{n} t_i + \frac{m-1}{m} t_n$$

Wir haben die untere Schranke $F^*(I) \geq \frac{1}{m} \sum_{i=1}^{n} t_i$.

Behauptung 2: In einem optimalen Plan für I werden nicht mehr als zwei Aufgaben von einem Prozessor erledigt.

Daraus folgt $F(I) - F^*(I) \le \frac{m-1}{m} t_n$ oder

$$\frac{F(I) - F^*(I)}{F^*(I)} \le \frac{m-1}{m} \frac{t_n}{F^*(I)}.$$

Da das Theorem für I nicht gilt, erhalten wir

$$\frac{1}{3} - \frac{1}{3m} < \frac{m-1}{m} \frac{t_n}{F^*(I)},$$

woraus $F^*(I) < 3t_n$ folgt.

Damit ist Behauptung 2 bewiesen.

Wir sind im Beweis des Theorems nun so weit:

Falls das Theorem für eine Instanz I versagt, dann werden in einem optimalen Plan für I jedem Prozessor höchstens zwei Aufgaben zugeordnet.

Dies passiert aber nie, denn es gilt:

Behauptung 3: Falls ein optimaler Plan für I jedem Prozessor höchstens zwei Aufgaben zuordnet, dann ist ein LPT-Schedule für I ebenfalls optimal.

Beweis von Behauptung 3: Übungsaufgabe!

□

Ein PTAS für Scheduling

Folgendes Verfahren führt zu einem PTAS:

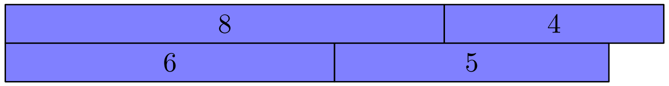
1. Wähle ein festes k (unabhängig von der Eingabe).
2. Berechne einen optimalen Plan für die k Aufgaben, mit der längsten Bearbeitungszeit.
3. Verplane die verbleibenden $n - k$ Aufgaben mit der LPT-Regel.

Warum sollte dieses Verfahren intuitiv gut sein?

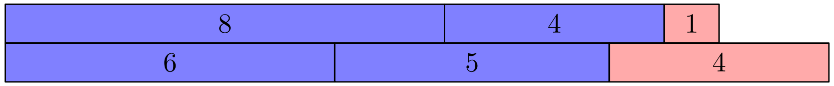
Beispiel

Sei $m = 2$, $n = 6$, $(t_1, \ldots, t_6) = (8, 6, 5, 4, 4, 1)$. Wir wählten $k = 4$.

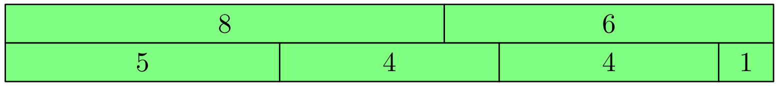
Schritt 1: Optimaler Plan für die 4 längsten Aufgaben.



Schritt 1: LPT-Plan für die verbleibenden 2 Aufgaben.



Die optimale Lösung ist:



Ein PTAS für Scheduling

Theorem

Sei $F(I)$ die Schlußzeit eines Plans, der durch das angebene Verfahrens erzeugt wird und $F^*(I)$ die optimale Schlußzeit von I.

Dann gilt

$$\frac{F(I) - F^*(I)}{F^*(I)} \leq \frac{1 - 1/m}{1 + \lfloor k/m \rfloor}.$$

Beweis

- Sei r die optimale Schlußzeit für die k längsten Aufgaben.
- Wir können $F(I) > r$ annehmen, denn sonst $F(I) = F^*(I)$.
- Sei wieder o.B.d.A. $t_1 \geq \cdots \geq t_n$.
- Wir nehmen $n > k$ an (sonst $F(I) = F^*(I)$).
- Wir nehmen $n > m$ an (sonst $F(I) = F^*(I)$).

Wir wählen ein $j > k$, so daß die Bearbeitung der Aufgabe j zum Zeitpunkt $F(I)$ endet.

Falls es so ein j nicht gibt, dann gilt schon $F(I) = F^*(I)$.

Gibt es aber so ein j, dann müssen alle Prozessoren bis zum Zeitpunkt $F(I) - t_j$ beschäftigt sein.

Wegen $t_{k+1} \geq t_j$ sind insbesondere alle Prozessoren auch bis zum (früheren) Zeitpunkt $F(I) - t_{k+1}$ beschäftigt.

Daher gilt

$$\sum_{i=1}^{n} t_i \geq m(F(I) - t_{k+1}) + t_{k+1},$$

denn alle Prozessoren sind je $F(I) - t_{k+1}$ Schritte beschäftigt und einer muß noch Aufgabe t_{k+1} danach erledigen.

Beweis (Fortsetzung)

Aus

$$\sum_{i=1}^{n} t_i \geq m(F(I) - t_{k+1}) + t_{k+1},$$

folgt

$$F^*(I) \geq \frac{1}{m} \sum_{i=1}^{n} t_i \geq F(I) - \frac{m-1}{m} t_{k+1}$$

und schließlich

$$F(I) - F^*(I) \leq \frac{m-1}{m} t_{k+1}. \qquad (*)$$

Um $\frac{F(I) - F^*(I)}{F^*(I)}$ abschätzen zu können, brauchen wir noch eine (gute) untere Schranke für $F^*(I)$.

Beweis (Fortsetzung)

Wir betrachten die ersten $k+1$ Aufgaben. Es gibt nur m Prozessoren.

Also muß es einen Prozessor geben, der mindestens $\lceil (k+1)/m \rceil = 1 + \lfloor k/m \rfloor$ von diesen $k+1$ Aufgaben allein erledigt.

Wegen $t_1 \geq \ldots \geq t_n$ dauert es mindesten t_{k+1} Schritte, jede dieser Aufgaben zu erledigen.

Es gibt also einen Prozessor, der $(1 + \lfloor k/m \rfloor) t_{k+1}$ Schritte lang beschäftigt ist.

$$F^*(I) \geq (1 + \lfloor k/m \rfloor) t_{k+1} \qquad (**)$$

Beweis (Fortsetzung)

$$F(I) - F^*(I) \leq \frac{m-1}{m} t_{k+1}. \qquad (*)$$

$$F^*(I) \geq (1 + \lfloor k/m \rfloor) t_{k+1} \qquad (**)$$

Kombinieren wir $(*)$ und $(**)$ erhalten wir

$$\frac{F(I) - F^*(I)}{F^*(I)} \geq \frac{(m-1)/m}{1 + \lfloor k/m \rfloor} = \frac{1 - 1/m}{1 + \lfloor k/m \rfloor}.$$

□

Ein PTAS für Scheduling

Um ein PTAS zu erhalten, müssen wir bedenken, daß ϵ Teil der Eingabe ist.

Daraus muß der Algorithmus ein geeignetes k berechnen, mit

$$\frac{1 - 1/m}{1 + \lfloor k/m \rfloor} \leq \epsilon.$$

Das funktioniert beispielsweise, wenn wir ein k wählen, daß grösser als $(m-1)/\epsilon - m$ ist.

Wie groß ist die Laufzeit? Sie ist $O(n \log n + m^k)$, wenn wir den exakten Teil mit einem Branch-and-Bound-Algorithmus lösen.

Das entspricht $O(n \log n + m^{(m-1)/\epsilon - m})$.

Für jedes feste m haben wir ein PTAS.

Frage: Was passiert wenn m nicht fest ist?

Entwurf eines FPTAS

Wir betrachten Optimierungsprobleme der Form

$$\begin{aligned} \text{maximiere } & \sum_{i=1}^{n} p_i x_i \\ \text{unter } & \sum_{i=1}^{n} a_{ij} x_i \leq b_j \text{ für } j = 1, \ldots, m \\ & x_i \in \{0,1\} \\ & p_i \geq 0 \\ & a_{ij} \geq 0 \end{aligned}$$

Ohne Beschränkung der Allgemeinheit können wir noch $a_{ij} \leq b_j$ für alle i und j annehmen, denn sonst muß ja $x_i = 0$ gelten.

Entwurf eines FPTAS

Definition

Für $1 \le k \le n$ nennen wir $x_i = y_i$, $i = 1, \ldots, k$ eine *zulässige Einschränkung*, wenn es wenigstens eine zulässige Lösung gibt, die mit $x_i = y_i$ für $i = 1, \ldots, k$ anfängt.

Wir nennen dann so eine zulässige Lösung eine *Vervollständigung* dieser zulässigen Einschränkung.

Seien nun $x_i = y_i$ und $x_i = z_i$ für $i = 1, \ldots, k$ zwei verschiedene zulässige Einschränkungen mit

$$\sum_{i=1}^{k} p_i y_i = \sum_{i=1}^{k} p_i z_i.$$

Wir sagen, daß $x_i = y_i$ die $x_i = z_i$ *dominiert*, falls es eine Vervollständigung von $x_i = y_i$ gibt, deren Wert der Zielfunktion mindestens so groß ist, wie die Zielfunktionswerte aller Vervollständigungen von $x_i = z_i$.

Wir betrachten Probleme, für welche es eine einfache Regel gibt, die Dominanz von zulässigen Einschränkungen entscheidet.

Beispiel

Knapsack: Gegeben n Gegenstände und zu jedem Gegenstand sein Wert und seine Größe. Ausserdem ist die Größe eines Rucksacks gegeben.

Ziel ist es, Gegenstände auszusuchen, die in den Rucksack gepackt werden sollen. Ihr Wert soll maximal sein, aber sie müssen alle hineinpassen.

Dieses Problem läßt sich leicht auf die gewünschte Art modellieren. Die Werte der Gegenstände sind die p_i und die a_{i1} sind ihre Größe.

Dominanz ist leicht zu entscheiden. Wie?

Entwurf eines FPTAS

Allgemein lassen sich diese Optimierungsprobleme exakt durch dynamisches Programmieren lösen:

Wenn bereits zulässige Einschränkungen S^i für $x_1, \ldots, x_i$ gegeben sind, können wir daraus alle zulässigen Einschränkungen für $x_1, \ldots, x_{i+1}$ berechnen. Aus dieser Menge können wir noch alle dominierten zulässigen Einschränkungen entfernen und erhalten dann S^{i+1}. Wir können mit dem leeren S^0 beginnen und eine optimale Lösung aus S^n entnehmen.

Problem: Von S^i nach S^{i+1} kann sich die Größe verdoppeln.

Wir betrachten verschiedene Methoden, die Größe von S^i polynomiell zu halten.

Runden

Da wir Dominanz verwenden, enthält S_i keine zwei zulässigen Einschränkungen mit gleichem Wert der Zielfunktion.

Gibt es also nur wenige mögliche, verschiedene Werte der Zielfunktion, dann bleibt S^i klein.

Eine grobe Idee besteht darin, eine Instanz I durch eine andere I' zu ersetzen, die sich nur durch die Koeffizienten p_i der Zielfunktion unterscheiden. Die neuen Koeffizienten q_i von I' sollen nur wenige verschiedene Werte der Zielfunktion erlauben.

Dabei haben I und I' immer noch dieselben zulässigen Lösungen. Ihre optimalen Lösungen können aber verschieden sein.

Wir müssen dafür sorgen, daß sie sich nicht zu sehr unterscheiden.

Beispiel

Sei $(p_1, \ldots, p_4) = (1.1, 2.1, 1001.6, 1002.3)$.

Die Zielfunktion nimmt folgende Werte in S^i an:

$$S^1: \{0, 1.1\}$$

$$S^2: \{0, 1.1, 2.1, 3.2\}$$

$$S^3: \{0, 1.1, 2.1, 3.2, 1001.6, 1002.7, 1003.7, 1004.8\}$$

$$S^4: \{0, 1.1, 2.1, 3.2, 1001.6, 1002.3, 1002.7, 1003.4, 1003.7, 1004.4, 1004.8, 1005.5, 2003.9, 2005, 2006, 2007.1\}$$

Insgesamt werden 31 zulässige Einschränkungen betrachtet. Ihre Zahl verdoppelt sich jeweils, da keine zwei vorkommen, für die die Zielfunktion übereinstimmt.

Wir ersetzen jetzt $(p_1, \ldots, p_4) = (1.1, 2.1, 1001.6, 1002.3)$ durch $(q_1, \ldots, q_4) = (0, 0, 1000, 1000)$.

Jetzt erhalten wir nur:

$$S^1: \{0\}$$

$$S^2: \{0\}$$

$$S^3: \{0, 1000\}$$

$$S^4: \{0, 1000, 2000\}$$

Der Wert der neuen Zielfunktion kann um höchstens 7.1 zu groß sein und der optimale Wert ist mindestens $\max\{p_i\} = 1002.3$. Das heißt $F^*(I) - F^*(I') \leq 7.1$ und $F^*(I) \geq 1002.3$. Wir erhalten

$$\frac{F^*(I) - F^*(I')}{F^*(I)} \leq 0.007.$$

Runden

Gegeben ein ϵ, müssen wir $q_1, \ldots, q_n$ so bestimmen, daß

$$\frac{F^*(I) - F^*(I')}{F^*(I)} \leq \epsilon$$

und

$$\sum_{i=1}^{n} |S^i| \leq u(n, 1/\epsilon)$$

gilt, wobei u ein Polynom ist.

Falls wir S^{i+1} aus S^i in Zeit polynomiell in $|S^i|$ berechnen können, dann haben wir ein FPTAS.

Runden

$$\frac{F^*(I) - F^*(I')}{F^*(I)} \leq \epsilon$$

können wir erzielen, durch

$$\sum_{i=1}^{n} |p_i - q_i| \leq \epsilon F^*(I). \qquad (*)$$

Wir definieren

$$q_i = \lfloor p_i/(\epsilon L/n) \rfloor \epsilon L/n.$$

Alle q_i sind dadurch ein Vielfaches von $\epsilon L/n$ und unterscheiden sich von p_i um höchstens $\epsilon L/n$. Das erfüllt $(*)$.

L ist eine geeignete untere Schranke für $F^*(I)$. Wir können immer $L := \max\{p_i\}$ wählen.

Runden

$$q_i = \lfloor p_i/(\epsilon L/n) \rfloor \epsilon L/n.$$

Alle q_i sind ein Vielfaches von $\epsilon L/n$.

Wie groß kann $|S^i|$ werden?

Wegen $p_j \leq L$, ist der Wert der Zielfunktion jeder zulässigen Einschränkung in S^i höchstens $i \cdot L$. Die Zielfunktionswerte sind Vielfache von $\epsilon L/n$.

Es gibt also höchstens $\left\lceil iL \middle/ \epsilon L/n \right\rceil = O(i \cdot n/\epsilon)$ verschiedene Elemente in S^I.

(Gleiche Zielfunktionswerte kommen nicht vor, wegen Dominanz.)

Runden

Wir erhalten

$$\sum_{i=1}^{n} |S^i| = O(n^3/\epsilon).$$

Das ergibt ein FPTAS in $O(n^3/\epsilon)$ Zeit unter folgenden Vorraussetzungen:

- Dominanz in linearer Zeit entscheidbar
- S^{i+1} aus S^i in $O(|S^i|)$ Schritten berechenbar
- Eine untere Schranke $L \geq \max\{p_i\}$
- Zum allgemeinen Schema passend

Beispiel

Wir haben einen Rucksack der Kapazität 1112 und Gegenstände deren Größe und Wert $1, 2, 10, 100, 1000$ ist. Wir wählen $\epsilon = 0.1$.

$L = \max\{p_i\} = 1000$. Wir runden auf Vielfache von $\epsilon L/n = 0.1 \cdot 1000/5 = 20$. Die q_i sind daher $0, 0, 0, 100, 1000$.

Wir erhalten
$S^5 = \{(0,0,0,0,0), (0,0,0,1,0), (0,0,0,0,1), (0,0,0,1,1)\}$.

Die optimale Lösung von I' ist $(0,0,0,1,1)$ mit Zielwert 1100.

Die eigentliche optimale Lösung ist 1112, da wir den Rucksack ganz füllen können.

Intervall-Partitionierung

Intervall-Partitionierung ist eine weitere Methode, um die Mengen S^i klein zu halten.

Sei P_i das Maximum von allen $\sum_{j=1}^n p_j x_j$, unter allen zulässigen Einschränkungen in S^i.

- Wir unterteilen das Intervall $[0, P_i]$ in Unterintervalle der Größe $\epsilon P_i/(n-1)$ (wobei das letzte kleiner sein kann).
- Von allen zulässigen Einschränkungen, deren $\sum_{j=1}^n p_j x_j$ in das gleiche Unterintervall fallen, eliminieren wir alle bis auf eine mit Dominanzregeln. Dabei tun wir so, als wären ihre Zielfunktionen gleich.
- Das S^{i+1} wird dann aus dem reduzierten S^i berechnet.

Intervall-Partitionierung

- Der Fehler, der beim Identifizieren aller zulässigen Einschränkungen in einem Unterintervall gemacht werden kann, ist höchstens $\epsilon P_i/(n-1)$.
- $P_i \le F^*(I)$, denn P_i ist kleiner oder gleich der Zielfunktion einer zulässigen Lösung.
- Fehler können sich fortpflanzen. Bei jedem Übergang $S^i \to S^{i+1}$ kann ein neuer Fehler hinzukommen.
- Der Gesamtfehler ist höchstens

$$F^*(I) - F(I) = \sum_{i=1}^{n-1} \frac{\epsilon P_i}{n-1} \le \epsilon F^*(I).$$

Intervall-Partitionierung

Die Anzahl der Unterintervalle ist höchstens $\lceil n/\epsilon \rceil + 1$.

Also ist $|S_i| = O(n/\epsilon)$ und

$$\sum_{i=1}^{n} |S_i| = O(n^2/\epsilon).$$

Unter guten Vorraussetzungen ergibt dies einen $O(n^2/\epsilon)$-Zeit-Algorithmus.

Ist eine bessere untere Schranke für $F^*(I)$ bekannt, so kann diese jederzeit statt P_i verwendet werden.

Beispiel

Wir haben wieder einen Rucksack der Kapazität 1112 und Gegenstände deren Größe und Wert $1, 2, 10, 100, 1000$ ist. Wir wählen $\epsilon = 0.1$.

$P_1 = 1$ und die Unterintervalle haben Länge $0.1/4 = 0.025$. Das ergibt $S^1 = \{(0), (1)\}$.

Daraus wird zunächst $S^2 = \{(0,0), (0,1), (1,0), (1,1)\}$ berechnet.

Die zugehörigen Werte der Zielfunktion sind $0, 2, 1, 3$. Das ergibt $P_2 = 3$ und Unterintervalle der Länge $0.1 \cdot 3/4 = 0.075$.

Die Intervalle sind $[0, 0.075), [0.075, 0.15), [0.225, 0.3), \ldots$. Es findet keine Reduzierung statt, da nur ein Wert in jedes Unterintervall fällt.

Als nächstes wird $S^3 = \{(0,0,0),(0,0,1),\ldots,(1,1,1)\}$ bestimmt. Die Werte sind jetzt $0, 10, 2, 12, 1, 11, 3, 13$.

Die Unterintervalle haben Länge $0.1 \cdot 13/4 = 0.325$. Wieder findet keine Reduktion statt.

Beim nächsten Schritt haben wir eine Länge von $0.1 \cdot 113/4 = 2.825$. Jetzt fallen beispielsweise $0, 1, 2$ in das gleiche Intervall.

Hätten wir die Gegenstände in umgekehrter Reihenfolge betrachtet, wäre es einfacher gewesen.

Wir können aber auch eine eigene untere Schranke L statt der P_i verwenden.

$L = \max\{p_i\} = 1000$. Die erste Intervalllänge ist daher bereits $0.1 \cdot 1000/4 = 25$.

Jetzt sind die Werte der Zielfunktion für S^1, S^2 und S^3 nur 0, für S^4 sind es 0 und 100 und für S^5 schließlich 0, 100, 1000 und 1100.

Als beste Lösung wird $(0,0,0,1,1)$ bestimmt.

Separation

Die Intervall-Partitionierung hat $[0, P_i]$ stur in Unterintervalle geteilt. Nur in gleich Unterintervalle fallende zulässige Einschränkungen werden zusammengefasst.

Separation geht ähnlich vor, teilt aber nicht in Unterintervalle ein.

Stattdessen werden zulässige Einschränkungen als gleichwertig angesehen, wenn sich ihre Werte um höchstens $\epsilon P_i/(n-1)$ unterscheiden.

Auch bei diesem Vorgehen kann der hinzugefügte Fehler nur $\epsilon P_i/(n-1)$ sein.

Separation

Sind die Werte $3.9, 4.1, 7.9, 8.1, 11.9, 12.1$ dann führt eine Intervalllänge von 2 zu keiner Reduktion.

Separation halbiert allerdings die Anzahl der Werte.

Intervall-Partitionierung kann aber ebenfalls besser sein:

Wir betrachten hierzu eine Knapsack-Instanz mit $(p_1,\ldots,p_5) = (w_1,\ldots,w_5) = (3, 1, 5.1, 5.1, 5.1)$. Es sei eine untere Schranke bekannt, so daß

$$\epsilon L/(n-1) = 2.$$

Was ergibt sich nun bei Intervall-Partitionierung und was bei Separation?

Nichtapproximierbare Probleme

Wir betrachten das folgende Optimierungsproblem:

$$\text{Maximiere } \sum_{i=1}^{n} a_i x_i$$
$$\text{unter } \sum_{i=1}^{n} a_i x_i \leq m$$
$$x_1, \ldots, x_n \in \{0, 1\}$$
$$\text{mit } m = \frac{1}{2} \sum_{i=1}^{n} a_i$$

Die optimale Lösung ist m, falls sich $\{a_1, \ldots, a_n\}$ in zwei Mengen mit gleicher Summe partitionieren läßt.

Dies ist ein NP-hartes Problem.

Nichtapproximierbare Probleme

Betrachte nun folgende Variante:

$$\text{Minimiere } 1 + k\left(m - \sum_{i=1}^{n} a_i x_i\right) \text{ unter } \sum_{i=1}^{n} a_i x_i \leq m$$
$$x_1, \ldots, x_n \in \{0, 1\}$$

Die optimale Lösung ist jetzt 1, falls sich $\{a_1, \ldots, a_n\}$ in zwei Mengen mit gleicher Summe partitionieren läßt.

Andernfalls ist sie mindestens $1 + k$.

Angenommen es gäbe einen ϵ-Approximationsalgorithmus.

Wähle $k > \epsilon$. Dann lößt der Approximationsalgorithmus das Problem exakt, falls die optimale Lösung m ist.

Dieses Problem ist für kein $\epsilon > 0$ approximierbar, solange $P \neq NP$.