Flüsse und Lineare Programmierung

Flüsse und Flussnetzwerke

Flüsse in Netzwerken haben viele wichtige Anwendungen:

- Entwurf von Bewässerungsanalangen, Kanalisationen, etc.,
- Modellierung von elektrischen Flüssen und Stromnetzwerken,
- Analyse von Waren- oder Geldströmen in Märkten,
- Verkehrsflüsse in Strassen- und Schienennetzen,
- und viele mehr...

Wir behandeln ein mathematisches Modell für Flussnetzwerke und entwickeln effiziente Algorithmen zur Berechnung von Flüssen.

Diese Algorithmen haben auch viele interessante Auswirkungen auf die Lösung anderer Optimierungsprobleme.

Flüsse und Flussnetzwerke

Ein Flussnetzwerk $G = (V, E, c, s, t)$ ist gegeben durch

- Knotenmenge V und Menge gerichteter Kanten E,
- Quelle s, Senke t,
- jede Kante $e \in E$ hat eine Kapazität $c(e) \geq 0$.

Ein Fluss ist eine Funktion $f : E \to \mathbb{R}$ wie folgt:

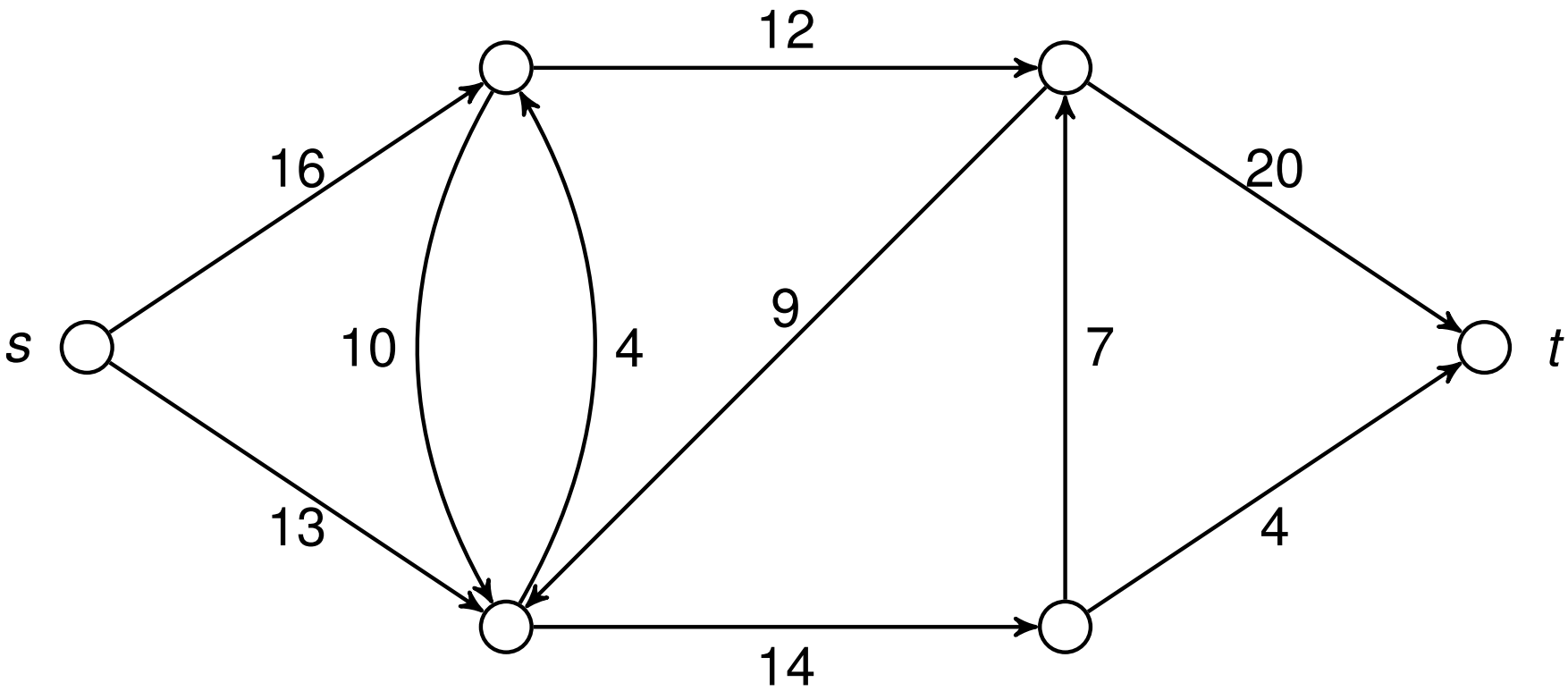
(1) Für jede Kante $e \in E$

$$0 \leq f(e) \leq c(e) \qquad \text{(Kapazitätsbeschränkung)}$$

(2) Für jeden Knoten $v \in V$ und $v \notin \{s, t\}$

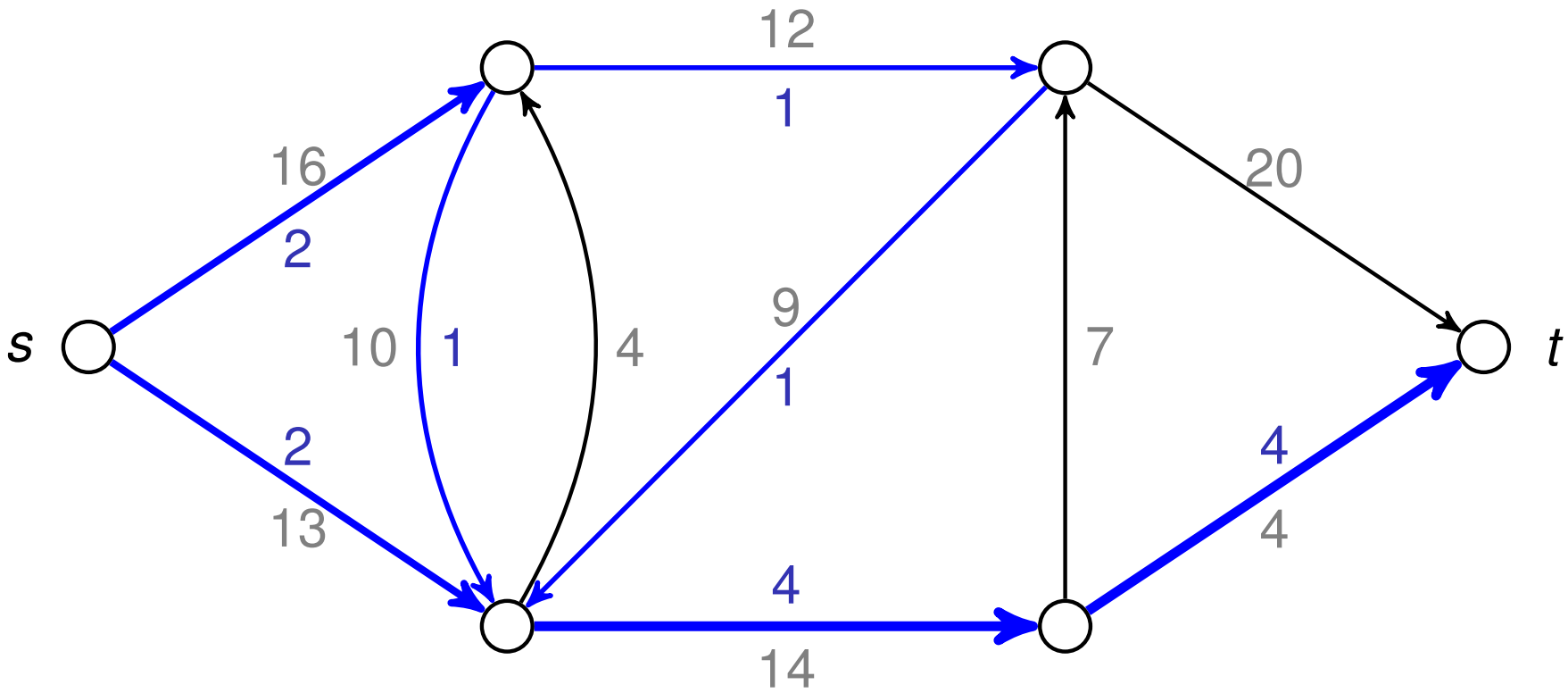
$$\sum_{(x,v) \in E} f((x, v)) = \sum_{(v,y) \in E} f((v, y)) \qquad \text{(Flusserhaltung bei } v\text{)}$$

Beispiel



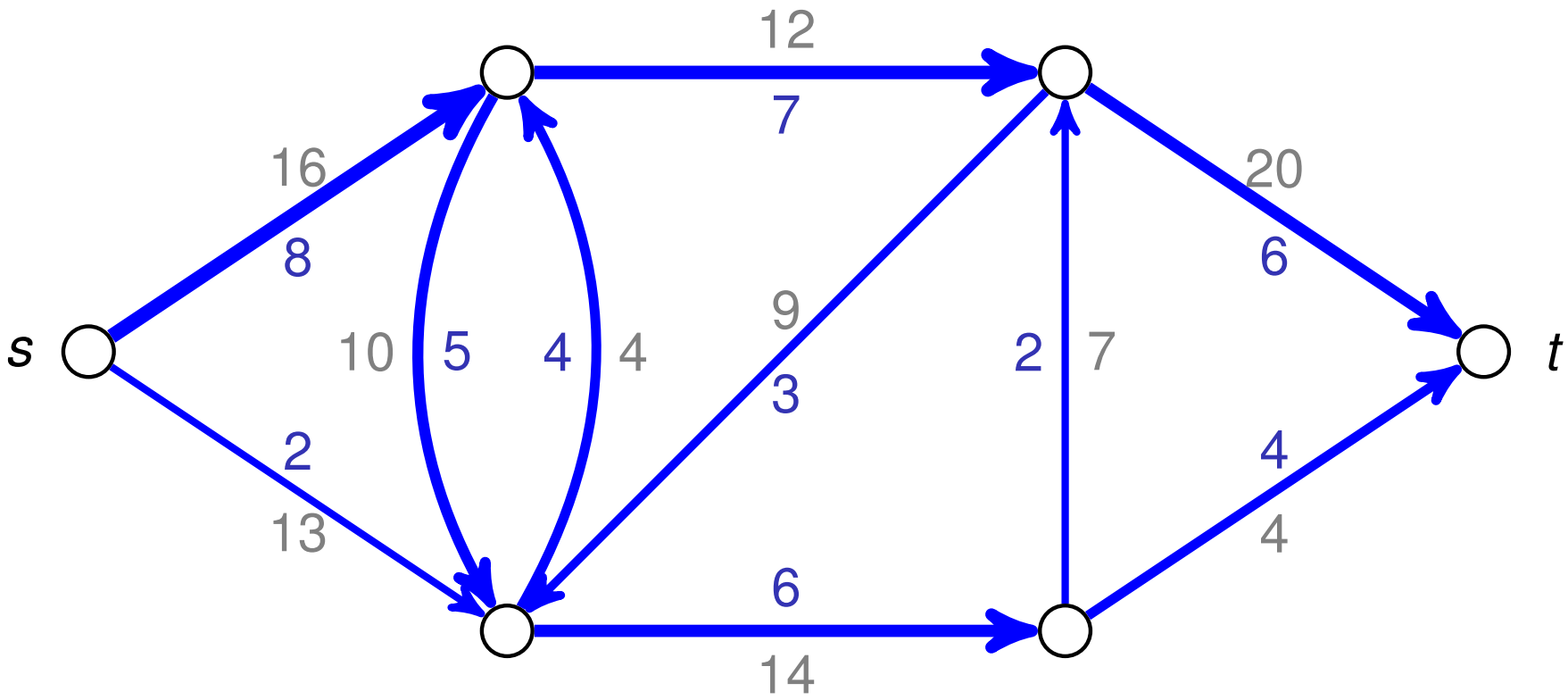
Ein Flussnetzwerk

Beispiel



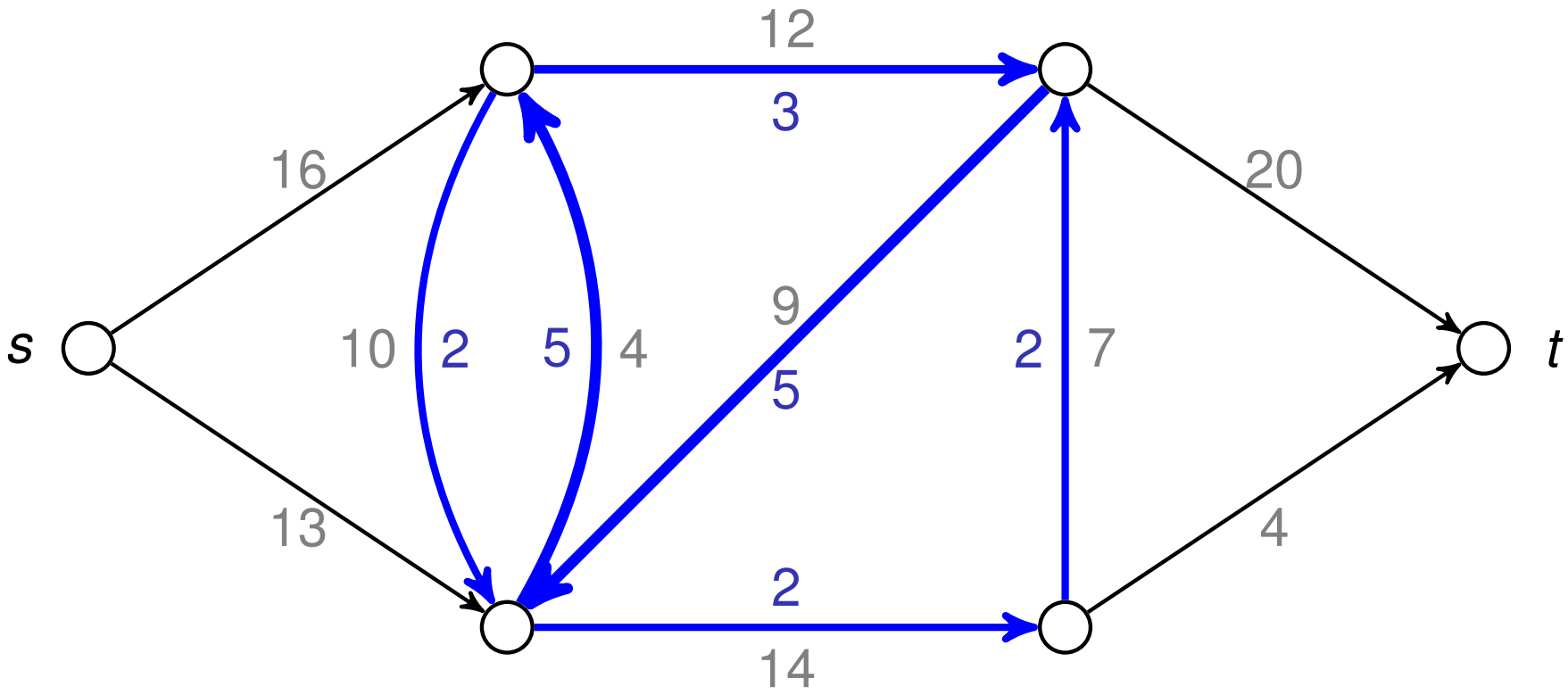
Ein möglicher Fluss

Beispiel



Ein anderer möglicher Fluss

Beispiel demogr.



Ist dies ein Fluss?

Auflösung: Nein, Kapazität verletzt.

Eine andere Sichtweise

Ein Fluss ist eine Funktion $f : V \times V \to \mathbb{R}$ definiert für alle geordneten Knotenpaare wie folgt:

(1) Für jede Kante $(u, v) \in E$ ist

$$f(u, v) \leq c((u, v))$$

und $f(u, v) \leq 0$ wenn $(u, v) \notin E$. (Kapazitätsbeschränkung)

(2) $f(u, v) = -f(v, u)$ (Symmetrie)

(3) Für jeden Knoten $v \in V$ und $v \notin \{s, t\}$

$$\sum_{u \in V} f(u, v) = 0 \qquad \text{(Flusserhaltung bei } v\text{)}$$

Beachte: $f(u, v) < 0$ ist erlaubt. Dann muss aber $f(v, u) > 0$ sein! Positiver Fluss von v nach u nur wenn Kante $(v, u) \in E$ und genug Kapazität $f(v, u) \leq c((v, u))$.

Wert eines Flusses

Gegeben einen Fluss f, definieren wir ...

- den Fluss zwischen Knotenmengen $U, W \subseteq V$:

$$f(U, W) = \sum_{u \in U} \sum_{w \in W} f(u, w)$$

- den Wert des Flusses

$$|f| := f(\{s\}, V) = f(V, \{t\})$$

$|f|$ ist die Menge an Fluss, die s verlässt und nicht zurückkehrt.

Ein **maximaler Fluss** f^* maximiert den Wert $|f^*|$.
Können wir f^* effizient berechnen?

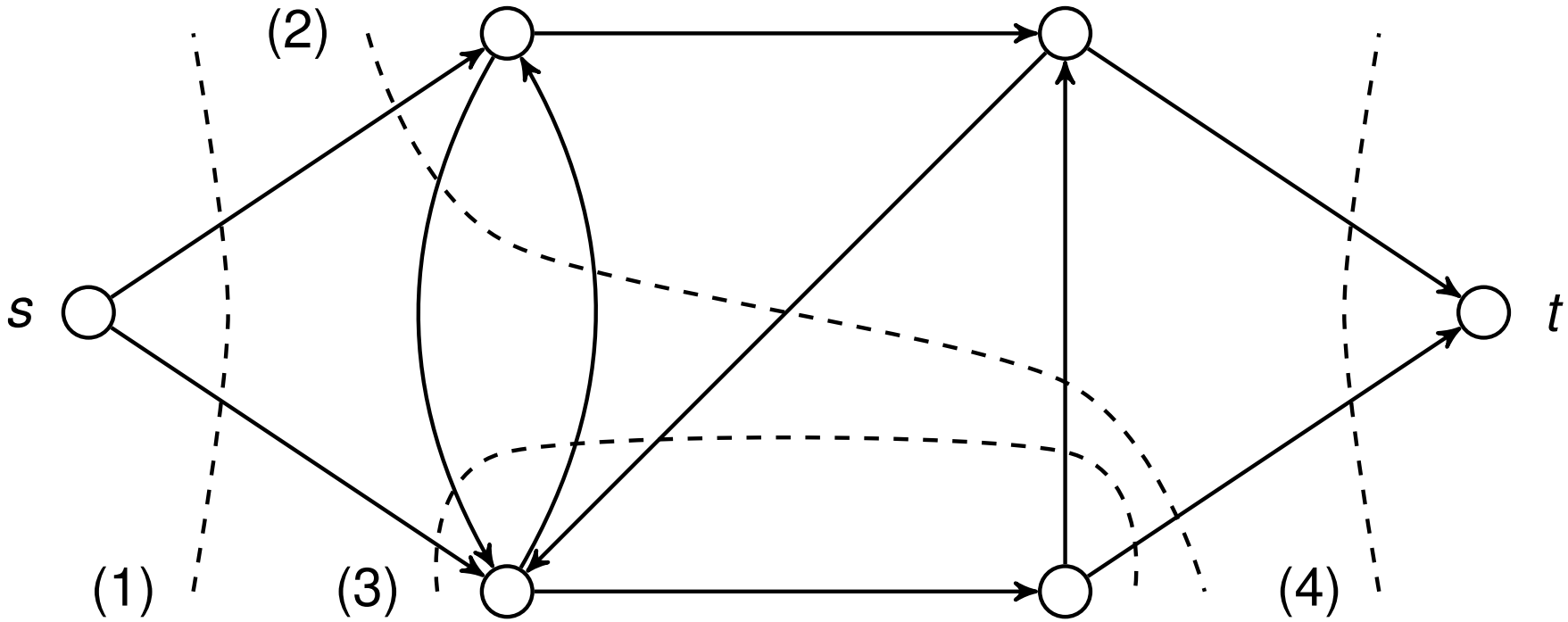
Schnitt im Graphen

Ein *s*-*t*-Schnitt in einem Graphen $G = (V, E)$ ist eine Aufteilung der Knoten in V in zwei Mengen S und T so dass $S \cup T = V$, $S \cap T = \emptyset$, sowie $s \in S$ und $t \in T$.

- Ein *s*-*t*-Schnitt (S, T) zerschneidet das Netzwerk in einen Teil mit Knotenmenge S (mit s) und den Rest $T = V \setminus S$ (mit t).
- Die Kapazität des Schnittes (S, T) ist gegeben durch

$$c(S, T) := \sum_{u \in S} \sum_{v \in T} c((u, v))$$

s-t-Schnitte multiple choice



Welche dieser Schnitte sind keine s-t-Schnitte?

Auflösung: (3)

Flüsse über s-t-Schnitte

Lemma

Es gilt für jeden Fluss f und jeden s-t-Schnitt, dass $f(S, T) = |f|$.

Beweis:

- Wir betrachten einen s-t-Schnitt $S = X \cup \{x\}$ und $T = Y$ mit $x \neq s$. Dann gilt

$$\begin{aligned} f(X \cup \{x\}, Y) &= f(X, Y) + f(x, Y) \\ &= f(X, Y) + f(x, Y) - \underbrace{(f(x, X) + f(x, Y))}_{= 0, \text{ Flusserhaltung}} \\ &= f(X, Y) - f(x, X) \\ &= f(X, Y) + f(X, x) = f(X, Y \cup \{x\}) \end{aligned}$$

Flüsse und Kapazitäten

- Über den Schnitt, der sich aus Verschiebung eines Knotens x in den anderen Teil ergibt, geht die gleiche Menge Fluss. Also geht über alle s-t-Schnitte die gleiche Menge Fluss.
- $(\{s\}, V \setminus \{s\})$ ist ein s-t-Schnitt mit Fluss $|f| = f(\{s\}, V \setminus \{s\})$. □

Für **jeden s-t-Schnitt** (S, T) und **jeden Fluss** f gilt:
$|f|$ ist beschränkt durch die Kapazität des Schnittes

$$|f| = f(S,T) = \sum_{u \in S} \sum_{v \in T} f(u,v) \leq \sum_{u \in S} \sum_{v \in T} c((u,v)) = c(S,T).$$

Flüsse und Kapazitäten

Ein sehr zentrales Resultat: (Beweis später)

Max-Flow-Min-Cut

Der Wert des maximalen Flusses ist **immer genau** die minimale Kapazität eines s-t-Schnittes

$$\max_{f \text{ Fluss}} |f| = \min_{(S,T)\ s\text{-}t\text{-Schnitt}} c(S,T)$$

Berechnung maximaler Flüsse

Idee

Die Idee: Solange ein Pfad P von s nach t existiert, über den man $|f|$ erhöhen kann, erhöhe den Fluss entlang P so viel wie möglich.

Zur formalen Umsetzung definieren wir ein angepasstes Netzwerk.

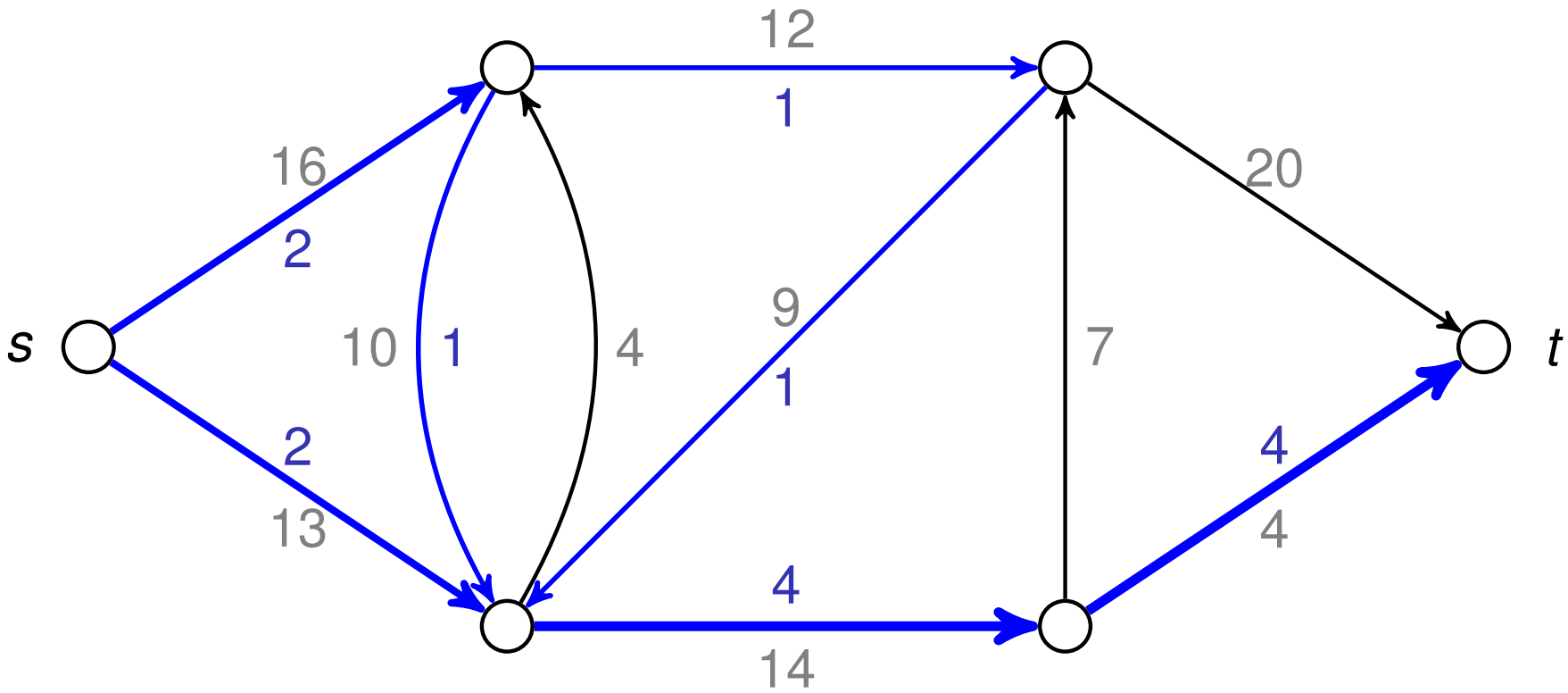
- Für Fluss f hat das Restnetzwerk G_f die Restkapazitäten

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

  G_f enthält alle Kanten (u, v) mit $c_f(u, v) > 0$.

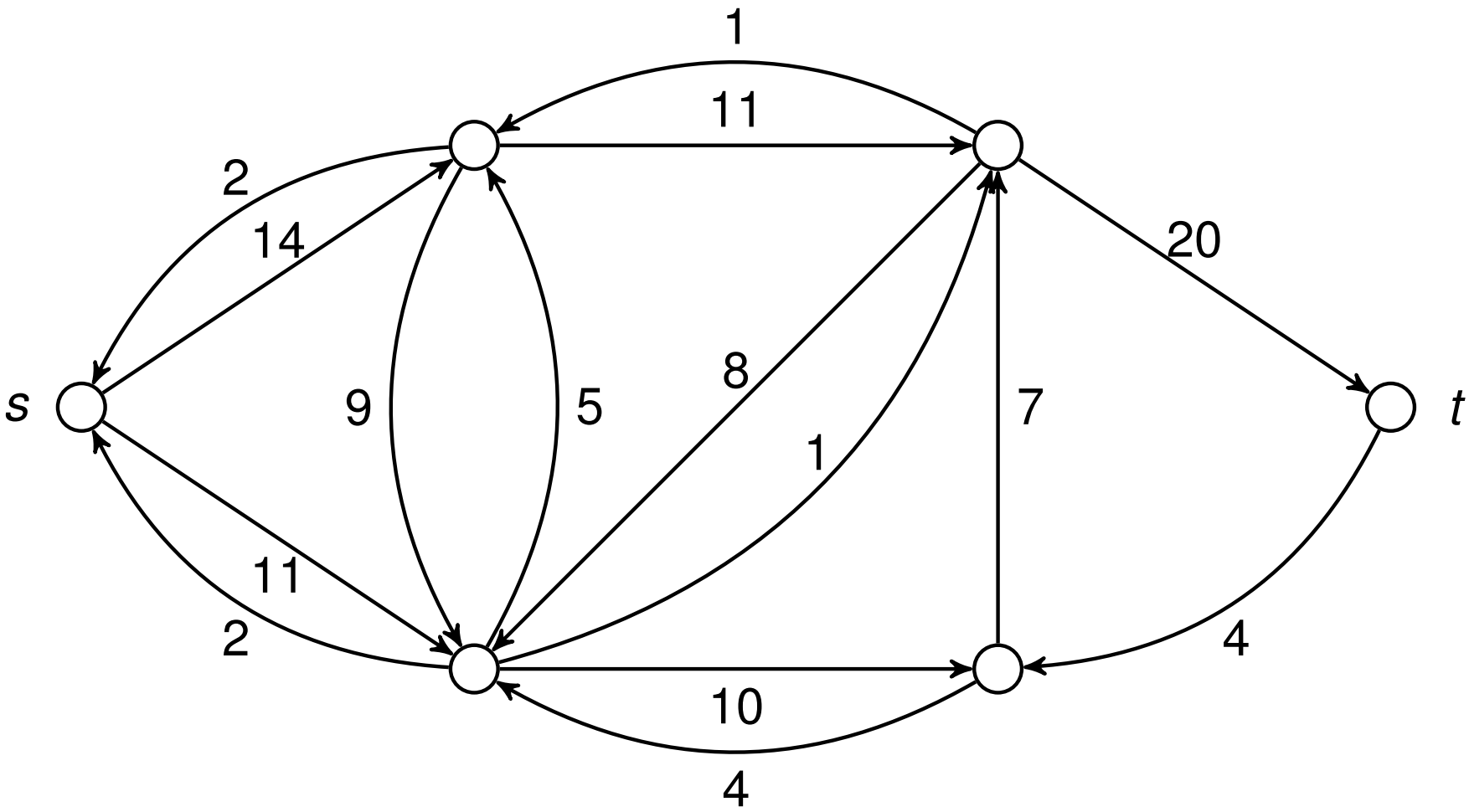
- Beachte, dass $c_f(u, v) \geq 0$ für alle $u, v \in V$.
- G_f enthält alle Knotenpaare (u, v), so dass der Fluss von u nach v erhöht werden kann – auch solche, wo nur Fluss von v nach u verringert werden kann!

Beispiel



Ein möglicher Fluss

Beispiel



Das Restnetzwerk für diesen Fluss

Flüsse und Restkapazitäten multiple choice

Sei f ein Fluss in G und f' ein Fluss in G_f. Was stimmt?

- (1) $f + f'$ ist ein Fluss in G
- (2) $f + f'$ ist ein Fluss in G_f
- (3) $f' - f$ ist ein Fluss in G_f
- (4) Wenn f maximal in G ist, dann ist $|f'| = 0$.
- (5) Wenn f' maximal in G_f ist, dann ist $|f| = 0$.
- (6) Wenn f' maximal in G_f ist, dann ist $f + f'$ maximal in G.

Auflösung: (1), (4), (6)

Ford-Fulkerson Algorithmus

(1) Anfangs setze $f(u, v) = 0$ für alle $u, v \in V$.

(2) Solange es einen s-t-Pfad P in G_f gibt:

(3) Erhöhe f entlang P so viel wie möglich:

- Sei $P = (v_1, \ldots, v_{\ell+1})$ der Pfad (mit $s = v_1$ und $t = v_{\ell+1}$)
- $\mu = \min\{c_f((v_i, v_{i+1})) \mid i = 1, \ldots, \ell\}$ ist die kleinste Restkapazität entlang P
- Definiere $f_P(v_i, v_{i+1}) = \mu$, $f_P(v_{i+1}, v_i) = -\mu$ für $i = 1, \ldots, \ell$ und $f_P(u, v) = 0$ sonst.
- Erhöhung: $f \leftarrow f + f_P$

Der s-t-Pfad P in G_f heißt auch augmentierender Pfad.

Ford-Fulkerson Algorithmus

- Terminiert der Algorithmus?
  - Allgemein nicht unbedingt (für reellwertige Kapazitäten).
  - Für rationalwertige Kapazitäten immer.
  - Für ganzzahlige Kapazitäten ist der Fluss immer ganzzahlig (Erhöhung immer um Integerwerte).
- Berechnet der Algorithmus einen maximalen Fluss? Ja!
- Wieviele Iterationen bei ganzzahligen Kapazitäten?
  - Eventuell $|f^*|$ viele – wenn immer ein P mit $f_P = 1$ gewählt wird.
- Wie sollten wir P wählen für wenige Iterationen?
  - Wähle P in G_f mit kleinster Anzahl von Kanten
  - Diese Variante heißt Edmonds-Karp Algorithmus.

Korrektheit und Max-Flow-Min-Cut

Max-Flow-Min-Cut

Die folgenden Aussagen sind äquivalent:

(1) f ist ein maximaler Fluss.

(2) f erlaubt keinen augmentierenden Pfad in G_f.

(3) Es gibt einen s-t-Schnitt (S, T) mit $f(S, T) = c(S, T)$.

Aus (1)+(2) folgt:
Der Ford-Fulkerson Algorithmus berechnet einen maximalen Fluss.

Beweis:

(3) $\Rightarrow$ (1): Klar, da $|f| \leq c(S, T)$ für alle Flüsse und s-t-Schnitte.

(1) $\Rightarrow$ (2): Klar, sonst erzeugt der Algorithmus einen besseren Fluss.

Beweis

(2) $\Rightarrow$ (3):

- Wenn (2) gilt, dann ist s in G_f nicht mit t verbunden.
- Für jeden s-t-Pfad Q in G gibt es eine (erste) Kante $(u, v) \in Q$, die nicht in G_f ist.
- Für diese Kante ist $c_f(u, v) = 0 = c(u, v) - f(u, v)$, also

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

- Sei nun

$$S = \{v \in V \mid \text{es gibt } s\text{-}v\text{-Pfad in } G_f\} \quad \text{und} \quad T = V \setminus S.$$

Dann ist (S, T) ein s-t-Schnitt und $|f| = f(S, T) = c(S, T)$. □

Edmonds-Karp Algorithmus

Wir wählen in jeder Iteration einen Pfad P mit kleinster Anzahl Kanten.

Lemma

Der Edmonds-Karp-Algorithmus berechnet einen maximalen Fluss in Laufzeit $O(|V| \cdot |E|^2)$.

Beweis:

- Wir zeigen zuerst, dass die Anzahl Kanten auf den kürztesten Weg von s nach v niemals sinken kann (für jedes $v \in V$).
- Finde einen kürzesten s-t-Pfad P in G_f mit Breitensuche von s.
  - Betrachte den Graphen G_f "top-down" von s
  - Breitensuche teilt V in Ebenen ein mit gleicher Distanz zu s
  - P ist ein kürzester s-t-Pfad, P verläuft nur "top-down" in G_f.

Edmonds-Karp Algorithmus

- Wir erhöhen nun den Fluss entlang P um f_P.
  - (Mindestens) eine Kante (u, v) in P mit kleinster Restkapazität hat $c_f(u, v) = f_P(u, v)$. Diese "top-down"-Kante verschwindet aus G_f
  - Kante (v, u) in Gegenrichtung bekommt mehr Restkapazität
  - Das kann die kürzesten Pfade von s nur verlängern.
- Phase k: Iterationen, in denen kürzester s-t-Pfad die Länge k hat.
  - Nach maximal $|E|$ Iterationen ist auf jedem kürzesten Pfad aus Phase k eine Kante "geflippt" worden.
  - Dann steigt die Distanz von s zu t um mindestens 1, und Phase k ist vorbei.
- Gesamtlaufzeit:
  - Es gibt höchstens $|V| - 1$ Phasen.
  - Jede Phase hat höchstens $|E|$ Iterationen.
  - Jede Breitensuche benötigt $O(|E|)$ Laufzeit. □

Bipartites Matching

Matchings

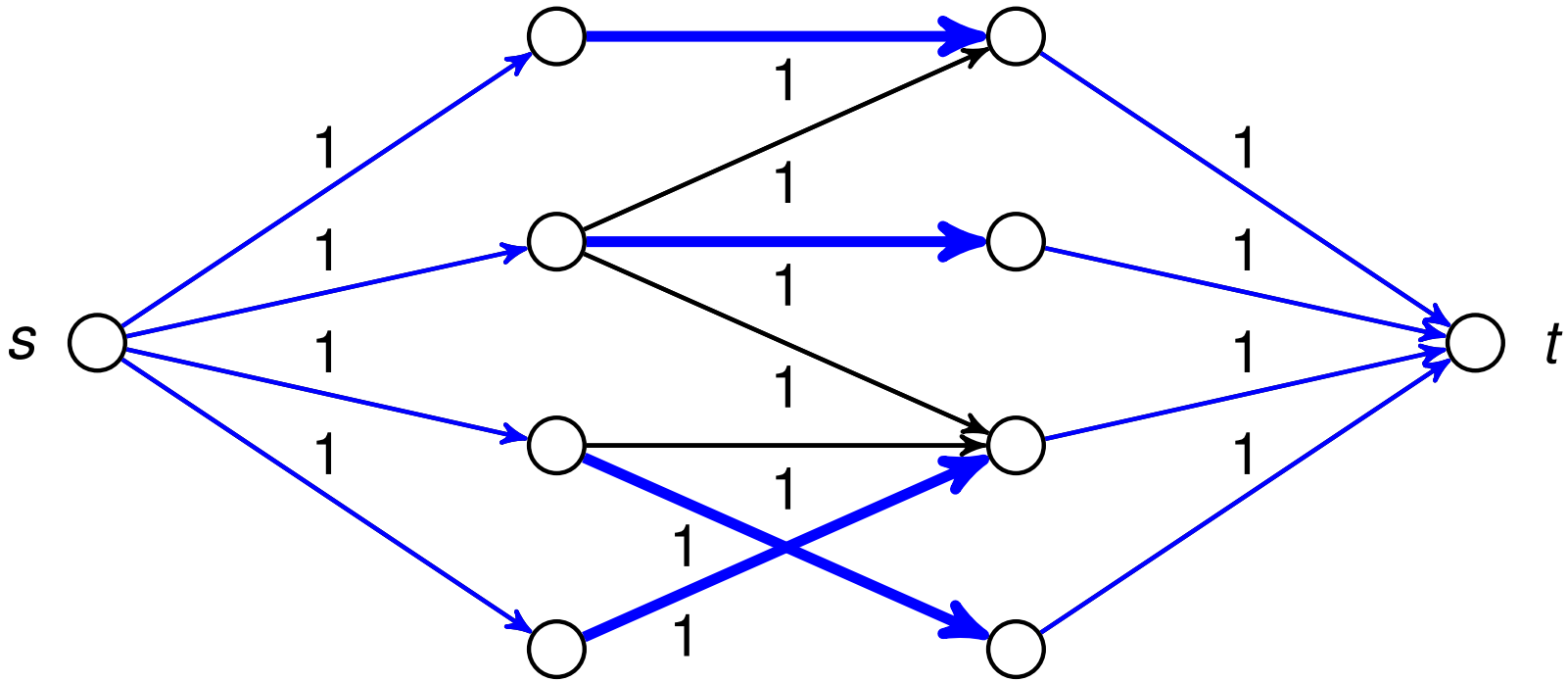
- Ein Matching $M \subseteq E$ in einem ungerichteten Graphen $G = (V, E)$ enthält für jeden Knoten $v \in V$ höchstens eine indzidente Kante.
- Wir sagen ...
  - Kante e ist gematcht ($e \in M$) oder ungematcht (sonst)
  - Knoten v ist gematcht (es gibt $\{u, v\} \in M$) oder ungematcht (sonst)
- Matching M heißt ...
  - perfekt: Jeder Knoten ist gematcht.
  - größtmöglich (engl.: maximum):
    Für jedes Matching M' gilt $|M| \geq |M'|$.
  - nicht-erweiterbar (engl: maximal):
    Für jedes $e \in E \setminus M$ gilt $M \cup \{e\}$ ist kein Matching.
- Perfekte Matchings $\subseteq$ Maximum Matchings $\subseteq$ Maximal Matchings

Maximum Matchings in bipartiten Graphen

Sei $G = (A \cup B, E)$ ein bipartiter Graph. Wir beschreiben eine Reduktion auf die Berechnung maximaler Flüsse:

- Erstelle ein (zuerst leeres) Flussnetzwerk G':
  - Füge alle Knoten aus $A \cup B$ hinzu.
  - Füge neue Quelle s und neue Senke t hinzu.
  - Füge eine gerichtete Kante (s, a) für jeden $a \in A$ hinzu.
  - Füge eine gerichtete Kante (b, t) für jeden $b \in B$ hinzu.
  - Füge eine gerichtete Kante (a, b) für jede $\{a, b\} \in E$ hinzu.
  - Alle Kanten erhalten Kapazität 1.
- Berechne einen maximalen Fluss f^* in G'.
- Sei $M = \{\{a, b\} \in E \mid f^*(a, b) = 1\}$ das berechnete Matching.

Maximum Matchings in bipartiten Graphen



Bipartities maximum Matching durch maximalen Fluss

Satz

Die Größe des maximum Matchings in G ist der Wert des maximalen Flusses in G'. Die Menge der Kanten, die Fluss von A nach B liefern, ist ein maximum Matching in G.

Beweis:

"$\Rightarrow$" Sei M ein maximum Matching. Erstelle einen Fluss f in G':

- Setze $f(s,a) = f(a,b) = f(b,t) = 1$ (sowie Gegenrichtung -1) für jede Kante $\{a,b\} \in M$.
- f ist s-t-Fluss mit $|f| = |M|$.
  Nur Kanten von M liefern Fluss von A nach B. $\surd$

Bipartites maximum Matching durch maximalen Fluss

"$\Leftarrow$" Sei f ein maximaler Fluss in G' mit binären Kantenflüssen. Betrachte das berechnete Matching M.

- $|f| = |M|$:
  - $S = A \cup \{s\}$ ist s-t-Schnitt in G'.
  - Also gilt $f(S, T) = f(A, B) = |f|$.
  - Da wir binäre Kantenflüsse haben, ist $|f| = |M|$.
- M ist Matching:
  - Höchstens eine Flusseinheit läuft bei $a \in A$ ein.
  - Höchstens eine Flusseinheit läuft bei $b \in B$ aus.
  - Flusserhaltung und binäre Flüsse $\Rightarrow$ für jeden Knoten $v \in A \cup B$ höchstens eine Kante $\{u, v\} \in M$. □

Laufzeit demogr.

In welcher Laufzeit berechnet der Ford-Fulkerson Algorithmus ein maximum Matching in einem bipartiten Graphen?

- (1) $\Theta(|V| + |E|)$
- (2) $\Theta(|V| \cdot |E|)$
- (3) $\Theta(|V| + |E| \log |E|)$
- (4) $\Theta(|V| \cdot |E|^2)$

Auflösung: (2) $\Theta(|V| \cdot |E|)$

Vertex Cover in bipartiten Graphen

Vertex Cover

Für einen ungerichteten Graphen $G = (V, E)$ ist eine Überdeckung eine Menge $\ddot{U}$ von Knoten, so dass alle Kanten einen Endpunkt in $\ddot{U}$ besitzen.

Jeder Knoten $v \in V$ erhält ein Gewicht $w_v \geq 0$. Gesucht ist eine Überdeckung mit minimalem Gesamtgewicht.

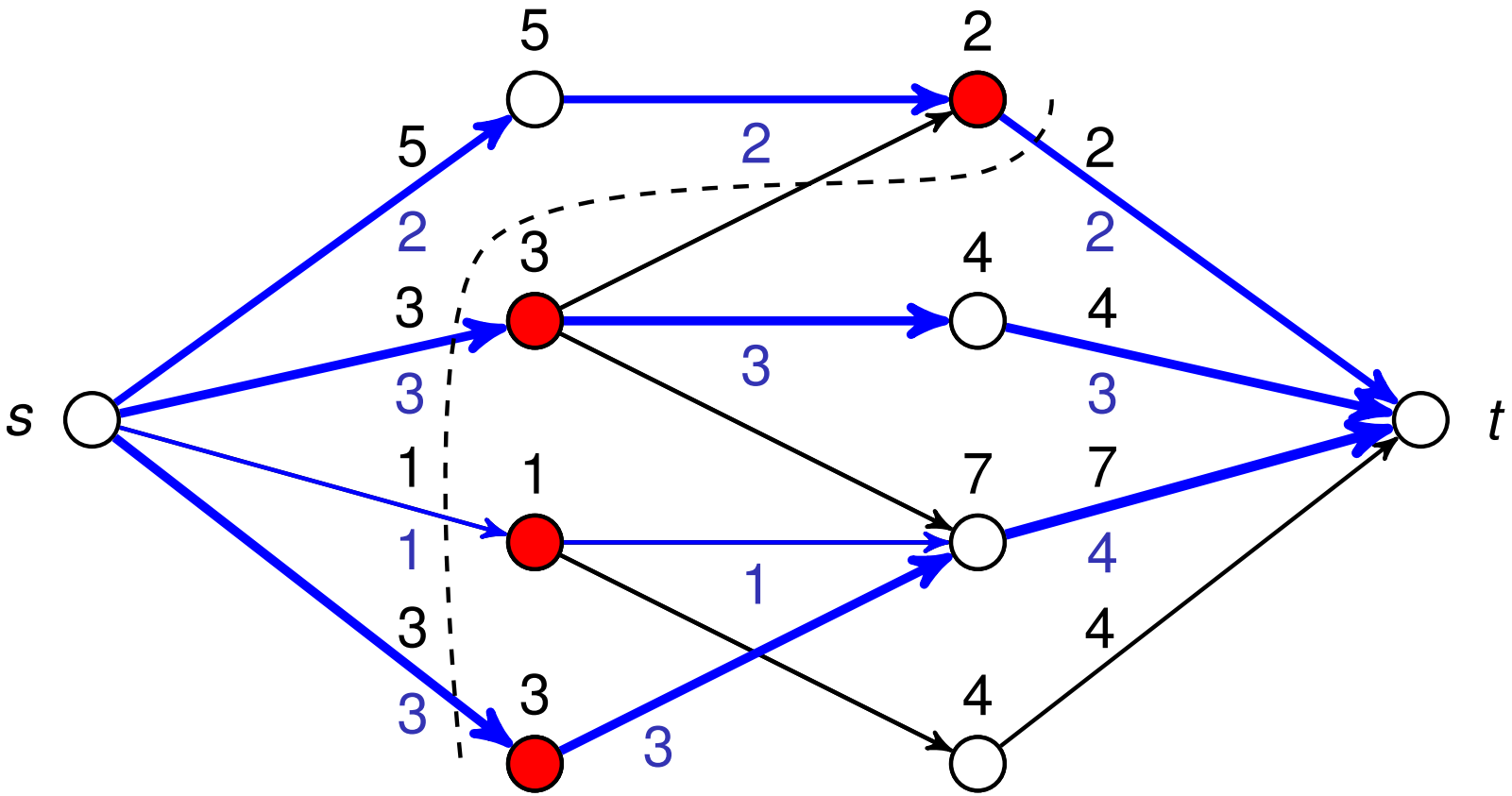
Wir lösen das Problem in **bipartiten Graphen** optimal mit Hilfe von maximalen Flüssen und minimalen Schnitten.

Vertex Cover in bipartiten Graphen

Sei $G = (A \cup B, E)$ ein bipartiter Graph. Wir beschreiben eine Reduktion auf die Berechnung minimaler Schnitte:

- Erstelle ein Flussnetzwerk G' (ähnlich wie für Matching):
  - Füge alle Knoten aus $A \cup B$ hinzu.
  - Füge neue Quelle s und neue Senke t hinzu.
  - Füge eine gerichtete Kante (s, a) für jeden $a \in A$ hinzu.
  - Füge eine gerichtete Kante (b, t) für jeden $b \in B$ hinzu.
  - Füge eine gerichtete Kante (a, b) für jede $\{a, b\} \in E$ hinzu.
- Die Kapazitäten sind diesmal anders:
  - Jede Kante (s, a) erhält Kapazität $c((s, a)) = w_a$.
  - Jede Kante (b, t) erhält Kapazität $c((b, t)) = w_b$.
  - Jede Kante (a, b) erhält Kapazität $c((a, b)) = \infty$.
- Berechne (max. Fluss f^* und) minimalen Schnitt $(S, T) \in G'$.
- Sei $\ddot{U} = (T \cap A) \cup (S \cap B)$ die berechnete Überdeckung

Beispiel



Korrektheit des Algorithmus

- Warum berechnet der Algorithmus eine Überdeckung?
  - Annahme: $\ddot{U}$ keine Überdeckung.
  - Dann gibt es $\{a, b\} \in E$ mit $a, b \notin \ddot{U}$, also $a \in S$ und $b \in T$.
  - Daraus folgt: $c(S, T) \geq c((a, b)) = \infty$
  - (S, T) ist kein minimaler Schnitt – Widerspruch.

- Warum ist die berechnete Überdeckung $\ddot{U}$ optimal?
  - Sei W eine bessere Überdeckung mit $\sum_{v \in W} w_v < \sum_{v \in \ddot{U}} w_v$.
  - Setze $S' = \{s\} \cup (A \setminus W) \cup (B \cap W)$ und $T' = V \setminus S'$.
  - Jede Kante ist überdeckt: $\{a, b\} \cap W \neq \emptyset$
    $a \in W \Rightarrow (s, a)$ im Schnitt. $a \notin W \Rightarrow b \in W$ und (b, t) im Schnitt.
  - Jeder s-t-Pfad nutzt eine Kante $\{a, b\} \in E$, aber dann ist (s, a) oder (b, t) im Schnitt. Also ist (S', T') ein s-t-Schnitt mit Kapazität

$$c(S', T') \leq \sum_{v \in W} w_v < \sum_{v \in \ddot{U}} w_v = c(S, T)$$

  - (S, T) ist kein minimaler Schnitt – Widerspruch.

Bipartites maximum Matching mit minimalen Kosten

Matching mit Kosten

Ein kleines Motivationsbeispiel:

- Wir haben verschiedene Fahrzeuge und möchten Kunden beliefern.
- Jedes Fahrzeug kann nur höchstens einen Kunden beliefern.
- Jedes Paar (Fahrzeug, Kunde) erzeugt evtl. unterschiedliche, nicht-negative Fahrtkosten.
- Manche Kunden können von einigen Fahrzeugen nicht beliefert werden, d.h. nicht alle (Fahrzeug, Kunde)-Paare sind erlaubt.

Das Ziel:
Beliefere die größtmögliche Anzahl Kunden (maximum Matching).
Unter den Lösungen, die die höchste Anzahl Kunden beliefern, wähle eine mit minimalen Gesamtfahrtkosten.

Perfektes Matching mit Kosten

- Für einen bipartiten Graphen $G = (A \cup B, E)$ und Kantenkosten $c(e) \geq 0$ für jede Kante $e \in E$ sind die Kosten eines Matchings M

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

- **Unser Ziel:**
  Berechne ein maximum Matching M mit kleinsten Kosten $c(M)$.
- Wir werden hier nur ein perfektes Matching mit minimalen Kosten berechnen (wenn es ein perfektes Matching gibt).
- Wenn wir das effizient berechnen können, können wir auch ein maximum Matching mit minimalen Kosten effizient berechnen (Übungsaufgabe).

Algorithmus

(1) Sei $G = (A \cup B, E)$ mit $k = |A| = |B|$

(2) Erstelle Flussnetzwerk G' für bipartites Matching

(3) $f = 0$, $M = \emptyset$, G_f Restnetzwerk von G' für f

(4) Solange $|f| < k$ und es gibt s-t-Pfad in G_f

(5) Setze Distanz für jede Kante $(u, v) \in E_f$

$$d((u,v)) = \begin{cases} c((u,v)) & \text{wenn } (u,v) \in E, \\ -c((v,u)) & \text{wenn } (u,v) \notin E \text{ aber } (v,u) \in E, \\ 0 & \text{wenn } \{u,v\} \cap \{s,t\} \neq \emptyset \end{cases}$$

(6) Sei P ein kürzester s-t-Pfad bzgl. d in G_f

(7) Erhöhe f entlang P so viel wie möglich

(8) Wenn $|f| = k$, gib $M = \{(a,b) \in E \mid f(a,b) = 1\}$ aus; sonst $\emptyset$.

Beobachtungen

- In jeder Iteration ein Fluss f mit binären Werten $f(e) \in \{0, 1, -1\}$ und ein entsprechendes Matching $M = \{(a, b) \in E \mid f(a, b) = 1\}$.
- Ab der zweiten Iteration ist ein augmentierender ein alternierender Pfad zwischen Kanten e mit $f(e) = 1$ ($e \in M$) und $f(e) = 0$ ($e \notin M$).
- f wird immer um 1 erhöht, genauso wie die Kardinalität von M.
- $d(P)$ seien die Distanzkosten von P. Sie entsprechen genau der Kostenänderung von M bei Erhöhung von f entlang P:
  - $c(e)$ für jede Kante $e \in P$, die in M eingefügt wird.
  - $-c(e)$ für jede Kante $e \in P$, die M verlässt.
- Für einen binären Fluss f sei M das entsprechende Matching. Wir bezeichnen mit G_M den Teil von G_f ohne Rückwärtskanten (a, s) oder (t, b).

Minimale Kosten und negative Kreise

Problem:
$d(e)$ hat negative Werte. Bei negativen Kreisen gäbe es evtl. keinen kürzesten s-t-Pfad. Wir sollten negative Kreise bzgl. d vermeiden!

In der optimalen Lösung ist das kein Problem (ohne Beweis).

Lemma

Sei M ein perfektes Matching. Dann gilt:
M hat minimale Kosten $\iff$ G_M hat keine negativen Kreise.

Wir werden nachweisen, dass in keiner einzigen Iteration ein negativer Kreis in G_M auftritt. Daraus folgt:

- Kürzeste Wege existieren und können effizient berechnet werden
- Perfektes Matching am Ende ist optimal (mit obigen Lemma)

Knotenpreise

- Wir nutzen eine Preisfunktion $h : V \to \mathbb{R}$ für die Knoten.
- Eine ökonomische Intuition
  - A seien Agenten, B seien Aufgaben
  - $d((a, b))$ sind die Kosten wenn Agent a Aufgabe b erledigt.
  - $h(a)$: “Rekrutierungskosten” wenn a (irgend-)eine Aufgabe macht
  - $h(b)$: “Ertrag” wenn jemand b erledigt.
- Die reduzierten Distanzen d^h von Kante $e = (u, v)$ bzgl. h sind

$$d^h(e) = h(u) + d(e) - h(v)$$

- **Achtung:** Diese Preise sind nur ein Gedankenexperiment um zu zeigen, dass keine negativen Kreise existieren!

Kompatible Preise

Eine Preisfunktion h heißt kompatibel mit Matching M wenn gilt

(1) für ungematchtes $a \in A$: $\quad h(a) = 0$.

(2) für jede Kante $\{a, b\} \in E$:

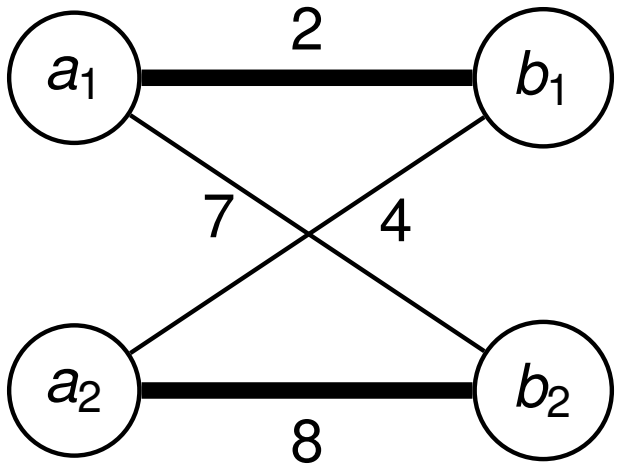
$$h(a) + d((a, b)) \geq h(b), \quad \text{also } d^h((a, b)) \geq 0,$$

(3) für jede Kante $\{a, b\} \in M$:

$$h(a) + d((a, b)) = h(b), \quad \text{also } d^h((a, b)) = 0.$$

- Kompatible Preise machen M billig:
  Es gilt $d^h(e) = 0$ für jede Kante $e \in M$ und $d^h(e) \geq 0$ für alle anderen Kanten!

Kompatible Preise multiple choice



Was sind kompatible Preise mit dem angezeigten Matching M?

(1) $h[a_1, b_1, a_2, b_2] = [0, 2, 0, 8]$

(2) $h[a_1, b_1, a_2, b_2] = [6, 8, 4, 12]$

(3) $h[a_1, b_1, a_2, b_2] = [3, 4, 0, 10]$

(4) $h[a_1, b_1, a_2, b_2] = [3, 5, 2, 10]$

Auflösung: (2), (4)

Kompatible Preise und negative Kreise

Lemma

Wenn M kompatible Preise hat, dann hat G_M keine negativen Kreise.

Beweis:

- Erweitere d^h auf die Kantenmenge E_M von G_M:
  $d^h(e) = h(u) + d(e) - h(v)$ für jede Kante $e \in E_M$
  $h(s) = 0, h(t) = \min_{e \in E} d(e)$
- Kompatible Preise bedeuten:
  - für $\{a, b\} \in M$ ist $(b, a) \in E_M$ und $d^h((b, a)) = -d^h((a, b)) = 0$.
  - für $\{a, b\} \in E \setminus M$ ist $(a, b) \in E_M$ und $d^h((a, b)) \geq 0$.
- Setze $h(s) \gg 0$ und $h(t) \ll 0$, so dass
  - $d^h((s, a)) = h(s) + 0 - h(a) \geq 0$ für jedes $a \in A$.
  - $d^h((b, t)) = h(b) + 0 - h(t) \geq 0$ für jedes $b \in B$.
- Damit werden alle reduzierten Kosten nicht-negativ!

Kompatible Preise und negative Kreise

- Dann gilt für jeden gerichteten Kreis C in G_M

$$d(C) = \sum_{e \in C} d(e) = \underbrace{\sum_{e \in C} d^h(e)}_{\text{Preise fallen weg}} \geq 0$$

□

Reduzierte Distanzen mit kompatiblen Preisen haben Vorteile:

- d^h sind nicht-negativ, nutze Dijkstra's Algorithmus!
- Für einen s-t-Pfad P sind es (fast) die echten Distanzen:

$$\begin{aligned} d^h(P) &= d^h((s,t)) + \sum_{j=1}^{\ell-1} d^h((v_j, v_{j+1})) + d^h((v_\ell, t)) \\ &= h(s) + \underbrace{\sum_{e \in P} d(e)}_{\text{Preise fallen weg}} - h(t) = d(P) + h(s) - h(t). \end{aligned}$$

Kürzeste Pfade mit kompatiblen Preisen

Berechne den kürzesten s-t-Pfad wie folgt:

- Finde optimalen s-t-Pfad P bzgl. d^h mit Aufruf von Dijkstra's Algorithmus
- Sei P' ein anderer s-t-Pfad. Es gilt

$$d(P') + h(s) - h(t) = d^h(P') \geq d^h(P) = d(P) + h(s) - h(t)$$

- P ist auch kürzester Pfad bzgl. originaler Distanzen.

Laufzeit

Für ein Matching M mit kompatiblen Preisen h kann der kürzeste s-t-Pfad bzgl. d in G_f mit einem Aufruf von Dijkstra's Algorithmus berechnet werden.

Preise aktualisieren

Wie aktualisieren wir die Preise, damit sie kompatibel bleiben?

- Sei M ein Matching mit kompatiblen Preisen h.
- Wir merken uns kürzeste-Wege-Distanzen von s zu allen anderen Knoten bzgl. d^h in G_M.
- Sei M' das Matching, das sich ergibt nach dem Update mit einem kürzesten s-t-Pfad P in G_M.
- Wir setzen neue Preise h' wie folgt:

$$h'(v) = h(v) + d^h(P_{s,v}) \qquad \text{für alle } v \in A \cup B \cup \{t\},$$

wobei $P_{s,v}$ ein kürzester s-v-Pfad in G_M bzgl. d^h war.

Lemma

Wenn h kompatibel mit M ist, dann ist h' ist kompatibel mit M'.

Es treten also **keine negativen Kreise** im Algorithmus auf.

Algorithmus v2.0, mit kompatiblen Preisen

(1) Setze $M = \emptyset$,
$h(a) = 0$ für alle $a \in A$, $h(b) = \min_{\{a,b\} \in E} c((a,b))$ für jedes $b \in B$
$h(s) = 0$, $h(t) = \min_{e \in E} c(e)$

(2) Solange M nicht perfekt und es s-t-Pfad in G_M gibt

- Finde kürzesten s-t-Pfad P in G_M bzgl. d^h
- Vergrößere M mit Hilfe von P
- Aktualisiere Preisfunktion mit kürzeste-Wege-Distanzen

(3) Wenn M perfekt, gib M aus; sonst $\emptyset$.

Laufzeit

Der Algorithmus berechnet ein perfektes Matching mit minimalen Kosten in Laufzeit $O(|V| \cdot T_{\text{Dijkstra}})$.

Kompatible Preise 2 multiple choice

Was stimmt?

(1) Kompatible Preise und reduzierte Distanzen sind notwendig, damit der Algorithmus das korrekte Ergebnis liefert.

(2) Kompatible Preise und reduzierte Distanzen verschnellern den Algorithmus durch Einsatz des Dijkstra-Verfahrens.

(3) Wenn M kompatible Preise hat, dann gibt es unendlich viele kompatible Preise für M.

(4) Wenn M kompatible Preise hat, dann sind die Preise eindeutig.

Auflösung: (2), (3)

Lineare Programmierung

Lineare Programmierung

Die Lineare Programmierung ist eine sehr mächtige Technik für die Lösung von Optimierungsproblemen.

- Bedingungen werden durch ein System von linearen Ungleichungen definiert.
- Eine optimale Lösung muss alle Ungleichungen erfüllen und eine lineare Zielfunktion optimieren.

Formal:

$$\text{Max.} \sum_{j=1}^{n} c_j \cdot x_j \quad \text{so dass} \quad \sum_{j=1}^{n} a_{i,j} \cdot x_j \leqslant b_i \text{ für } i = 1, \ldots, m$$
$$x_1, \ldots, x_n \geqslant 0.$$

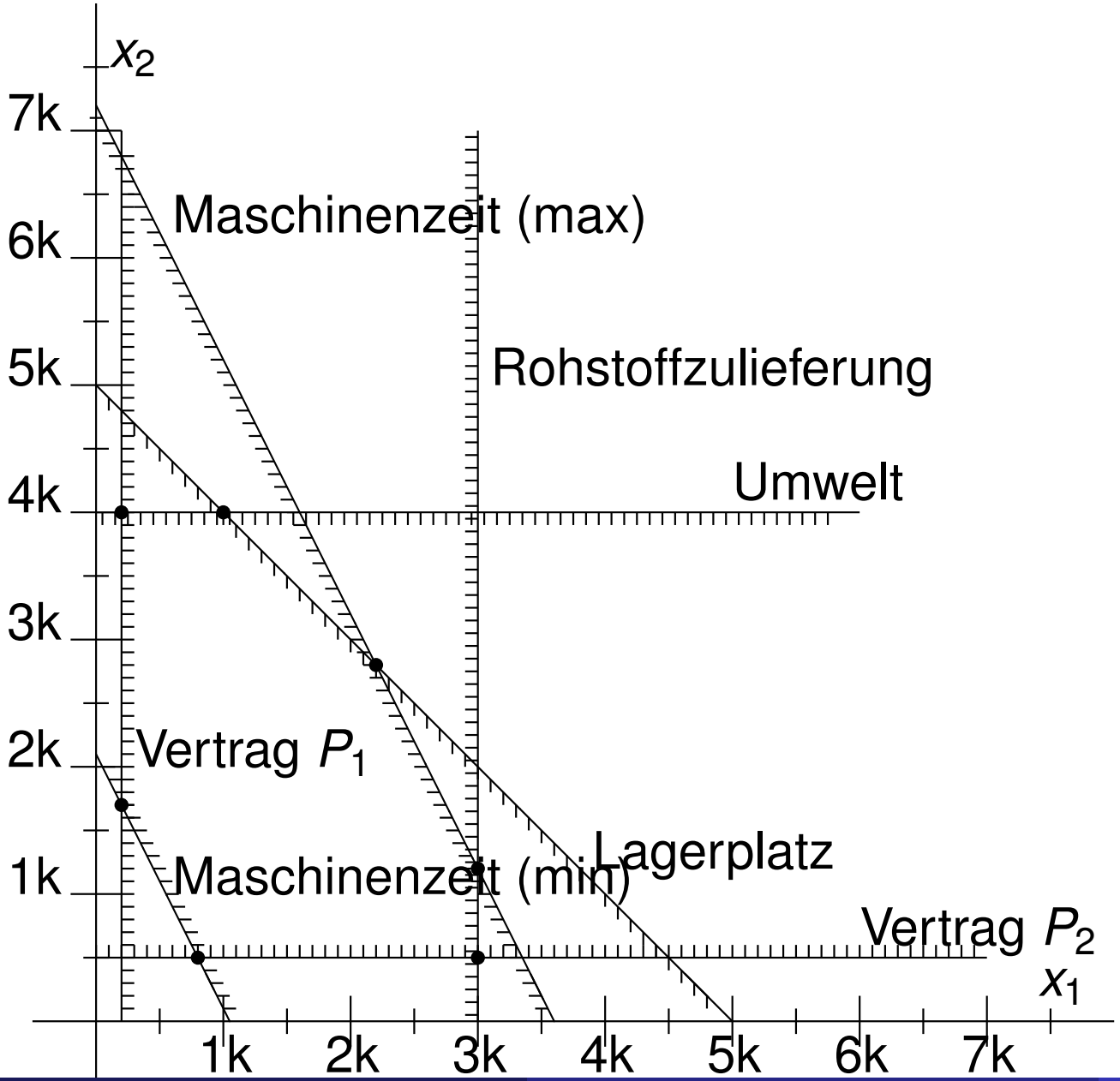
Ein Beispiel

- Ein Unternehmen verfügt über eine Maschine, auf der zwei verschiedene Produkte P_1 und P_2 produziert werden können.
- Eine Einheit von P_1 bringt 7 Euro Gewinn, eine Einheit P_2 bringt 4 Euro. Die Zielfunktion ist $7 \cdot x_1 + 4 \cdot x_2$.
- Die Produktion einer Einheit von P_1 nimmt die Maschine 4 Minuten in Anspruch, eine Einheit P_2 ist in 2 Minuten fertig. Die Maschine steht im Monat für 240 Stunden, also für 14400 Minuten, zur Verfügung. Wir erhalten die Ungleichung $4 \cdot x_1 + 2 \cdot x_2 \leqslant 14400$.
- Weitere Bedingungen wie Rohstoffverbrauch, Absatzmöglichkeiten etc. führen auf weitere Ungleichungen.

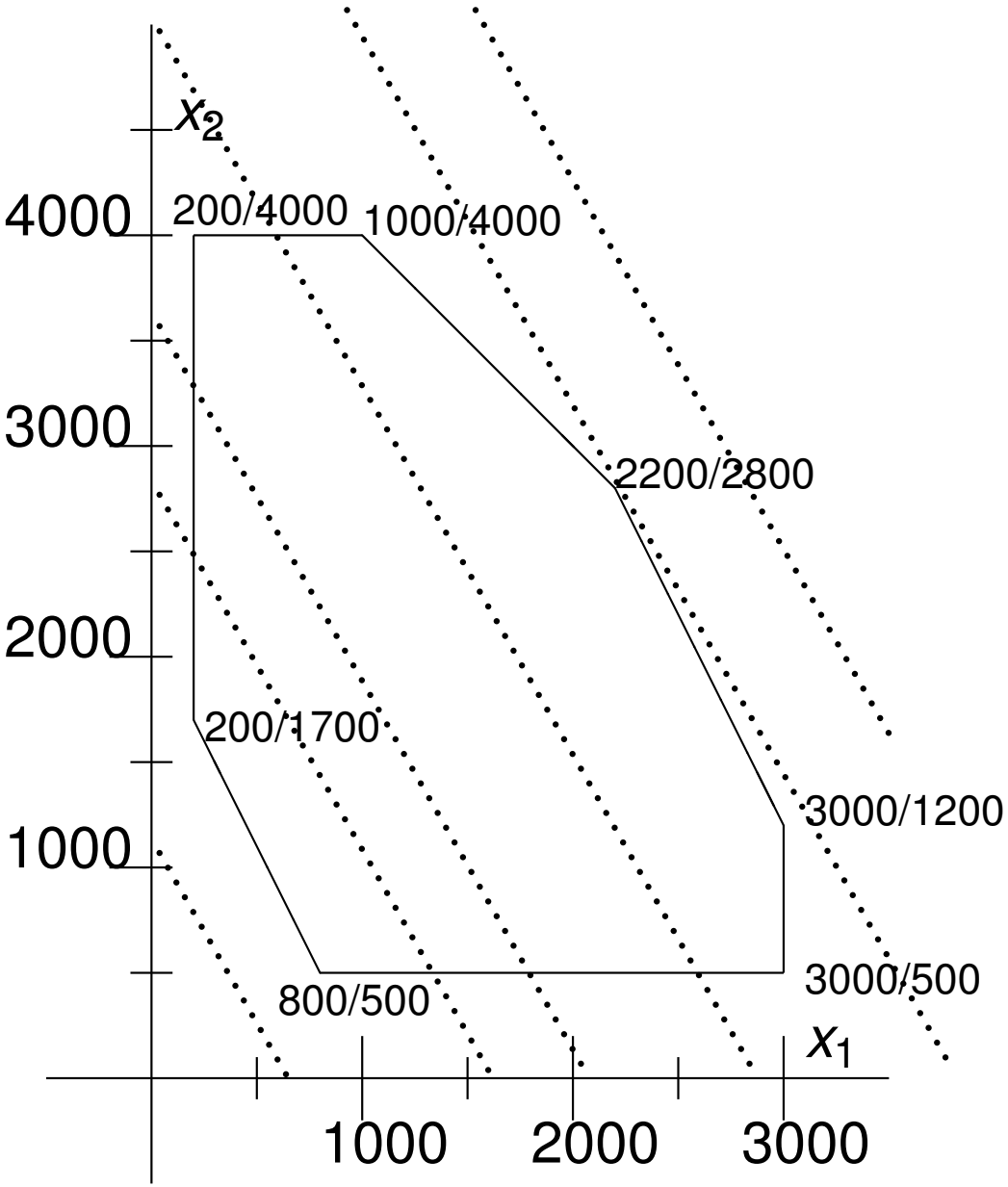
Wie kann eine optimale Lösung bestimmt werden?

Zuerst eine geometrische Interpretation.

Die Ungleichungen



Der Lösungsraum und die Zielfunktion



Effiziente Algorithmen für die Lineare Programmierung

Man bezeichnet den Lösungsraum auch als Polytop.

- Um eine optimale Lösung zu bestimmen, verschiebe die Gerade $7 \cdot x_1 + 4 \cdot x_2 = 0$ so weit wie möglich „nach oben“.
- Die Gerade wird dann auf eine Ecke treffen. (Ecken erfüllen zwei oder mehrere Ungleichungen exakt.)
- Die Situation im n-dimensionalen ist ungleich komplizierter, aber es gibt schnelle Algorithmen:
  - Der Simplex-Algorithmus bewegt sich von Ecke zu Ecke. In der Praxis extrem schnell, aber im worst case exponentielle Laufzeit.
  - Interior-Point Verfahren durchstossen das Innere des Polytops und garantieren Laufzeiten polynomiell in n,m und $\log \max_{i,j}\{c_i, b_j, a_{i,j}\}$.
  - **Große offene Frage:** Gibt es stark polynomielle Algorithmen für lineare Programmierung, d.h. mit polynomieller Laufzeit unabhängig von der Codierung der Zahlen?

Lineare Programme demogr.

Ist das ein zulässiges lineares Programm?

$$\begin{array}{lrcl} \text{Min.} & -7x_1 - x_2 - 5x_3 & & \\ \text{s.d.} & x_1 - 2x_2 + x_3 & \leq & 10 \\ & -2x_2 + x_3 & \geq & 5x_1 - 6 \\ & x_1, x_2, x_3 & \geq & 0 \end{array} \qquad \begin{array}{lrcl} \text{Max.} & 7x_1 + x_2 + 5x_3 & & \\ \text{s.d.} & x_1 - 2x_2 + x_3 & \leq & 10 \\ & 5x_1 + 2x_2 - x_3 & \leq & 6 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

- Ja.
- Nein.
- Keine Ahnung

Auflösung: Ja, negative Parameter erlaubt!

- Min. $\sum_j c_j x_j \iff$ Max. $\sum_j -c_j x_j$
- $\sum_j a_{i,j} x_j \leq b_i \iff \sum_j -a_{i,j} x_j \geq -b_i$

Lineare Nebenbedingungen demogr.

$$\begin{aligned} \text{Max.} \quad & \sum_{j=1}^{n} c_j \cdot x_j \\ \text{so dass} \quad & \sum_{j=1}^{n} a_{i,j} \cdot x_j \leqslant b_j \quad \text{für } i = 1, \ldots, m \\ & x_j \geq 0 \quad \text{für } i = 1, \ldots, n \end{aligned}$$

Kann man die Bedingung $x_2 = 2x_5$ in einem linearen Programm ausdrücken?

- Ja.
- Nein.
- Keine Ahnung

Auflösung: Ja, benutze $x_2 - 2x_5 \leq 0$ und $-x_2 + 2x_5 \leq 0$.

Beispiel: Maximale Flüsse

- Sei G ein Flussnetzwerk mit Kantenmenge E.
- Für Kante $(u, v) \in E$ sei $x_{u,v}$ der Fluss, der über die Kante fließt.

Formulierung als ein lineares Program (LP):

$$
\begin{aligned}
\text{Max.} \quad & \sum_{(s,v)\in E} x_{s,v} - \sum_{(u,s)\in E} x_{u,s} & & \\
\text{s.d.} \quad & \sum_{(u,v)\in E} x_{u,v} = \sum_{(v,w)\in E} x_{v,w} & & \text{für alle Knoten } v \neq s, t \\
& x_{u,v} \leq c((u,v)) & & \text{für alle Kanten } (u,v) \in E \\
& x_{u,v} \geq 0 & & \text{für alle Kanten } (u,v) \in E
\end{aligned}
$$

Beispiel: Bipartites Matching mit maximalem Gewicht

- Menge A von Angestellten, Menge B von Aufgaben. Kompetenz $k_{a,b} \geq 0$ des Angestellten a für die Aufgabe b.
- Weise jeder Aufgabe b höchstens einen Angestellten $a(b)$ und jedem Angestellten höchstens eine Aufgabe zu.
- Maximiere die Summe der Kompetenzen $\sum_{a \in A} k_{a(b),b}$.

Formulierung als ein lineares Program (LP):

$$
\begin{aligned}
\text{Max.} \quad & \sum_{a \in A, b \in B} k_{a,b} \cdot x_{a,b} & & \\
\text{s.d.} \quad & \sum_{a \in A} x_{a,b} & \leq \; 1 & \quad \text{für alle Aufgaben } b \in B \\
& \sum_{b \in B} x_{a,b} & \leq \; 1 & \quad \text{für alle Angestellten } a \in A \\
& x_{a,b} & \geq \; 0 & \quad \text{für alle } a \in A,\, b \in B
\end{aligned}
$$

Fraktional und Integral

- Aber wir suchen eine integrale Lösung, d.h., $x_{a,b} \in \{0, 1\}$ (statt einer fraktionalen Lösung $x_{a,b} \geq 0$).

  Nur dann beschreibt x ein legales Matching!

- Wie bekommen wir die beste integrale Lösung? Von selbst! Jede Ecke des LP ist integral. Es gibt immer ein integrales Optimum.

Satz

Jede Ecke x des Matching-Polytops ist integral. Es gibt immer eine integrale optimale Lösung für das Matching-LP.

Beispiel: Vertex Cover

- Das Matching-LP hat nur integrale Ecken. Das ist eine bemerkenswerte Ausnahme! Die allermeisten LPs haben nur fraktionale Optimallösungen.
- Hier ein LP für das (ungewichtete) Vertex Cover Problem in einem Graphen $G = (V, E)$:

Lineares Program (LP) für Vertex Cover:

$$\begin{aligned} \text{Min.} \quad & \sum_{v \in V} x_v & & & \\ \text{s.d.} \quad & x_u + x_v & \geq & 1 & \text{für alle } \{u, v\} \in E \\ & x_v & \geq & 0 & \text{für alle } v \in V \end{aligned}$$

Beispiel: Vertex Cover

- Eine legale Überdeckung enthält jeden Knoten nur ganz ($x_v = 1$) oder gar nicht ($x_v = 0$). Überdeckungen sind integrale Lösungen des LPs.
- Das LP hat aber evtl. nur fraktionale Optima mit $x_v^* \in (0, 1)$ (z.B. für vollständigen Graphen alle $x_v^* = \frac{1}{2}$).
- Bekommt man auch effizient eine optimale integrale Lösung? Nicht immer, das Problem ist NP-hart!
- Kann man das fraktionale Optimum in eine gute integrale Lösung "umrechnen"? Ja, 2-approximativ durch deterministisches Runden: $v \in \ddot{U} \iff x_v^* \geq \frac{1}{2}$.

Dualität

Obere Schranken für ein Lineares Programm

- LPs haben eine Dualitätseigenschaft und kommen in Paaren!
- Duales LP: Finde eine möglichst gute obere Schranke auf den Optimalwert des LPs.
- Wie finden wir eine gute obere Schranke auf den Optimalwert von

$$\begin{aligned} \text{Max.} \quad & 10x_1 + 6x_2 & & \\ \text{s.d.} \quad & x_1 + 5x_2 & \leq & 7 \\ & -x_1 + 2x_2 & \leq & 1 \\ & 3x_1 - 2x_2 & \leq & 5 \\ & x_1, x_2 & \geq & 0 \end{aligned}$$

- **Idee:** Nutze und kombiniere die Nebenbedingungen!

Obere Schranken

- Finde Faktoren y_1, y_2, y_3 für die Nebenbedingungen, so dass

$$\begin{aligned} 10x_1 + 6x_2 &\leq y_1(x_1 + 5x_2) + y_2(-x_1 + 2x_2) + y_3(3x_1 - 2x_2) \\ &= x_1(y_1 - y_2 + 3y_3) + x_2(5y_1 + 2y_2 - 2y_3) \\ &\leq 7y_1 + y_2 + 5y_3 \end{aligned}$$

- Die obere Schranke soll so klein wie möglich sein:

$$\text{Min.} \quad 7y_1 + y_2 + 5y_3$$

- Die erste Ungleichung gilt nur, wenn die Koeffizienten für x_1, x_2 rechts individuell größer sind:

$$\begin{aligned} y_1 - y_2 + 3y_3 &\geq 10 && \text{(für } x_1\text{)} \\ 5y_1 + 2y_2 - 2y_3 &\geq 6 && \text{(für } x_2\text{)} \end{aligned}$$

- Die zweite Ungleichung gilt nur, wenn $y_1, y_2, y_3 \geq 0$.

Das duale LP

- Die beste obere Schranke, die wir auf diese Art finden können, kann wieder durch ein LP formuliert werden:

$$\begin{aligned} \text{Min.} \quad & 7y_1 + y_2 + 5y_3 & & \\ \text{s.d.} \quad & y_1 - y_2 + 3x_3 & \geq & \ 10 \\ & 5y_1 + 2y_2 - 2y_3 & \geq & \ 6 \\ & y_1, y_2, y_3 & \geq & \ 0 \end{aligned}$$

- Wir nennen das originale das primale LP und dies das duale LP.
- Man kann das gleiche Verfahren für das duale LP anwenden – dann erhält man das primale LP zurück!
- Das duale LP des dualen LPs ist das primale LP!

Primales und duales LP sind dual zueinander.

Matrix-Vektor Notation

$$
\begin{aligned}
\text{Max.} \quad & \sum_{j=1}^{n} c_j \cdot x_j & & & \\
\text{s.d.} \quad & \sum_{j=1}^{n} a_{i,j} \cdot x_j & \leqslant & b_j & \text{für } i = 1, \ldots, m \\
& x_j & \geq & 0 & \text{für } i = 1, \ldots, n
\end{aligned}
$$

kann man viel kompakter schreiben als

$$
\begin{aligned}
\text{Max.} \quad & \mathbf{c}^T\mathbf{x} & & \\
\text{s.d.} \quad & \mathbf{Ax} & \leq & \mathbf{b} \\
& \mathbf{x} & \geq & \mathbf{0}
\end{aligned}
$$

- $\mathbf{A} = (a_{i,j})_{i,j}$ eine $(m \times n)$-Matrix
- $\mathbf{b} = (b_1, \ldots, b_m)^T$ Spaltenvektor.
- $\mathbf{c} = (c_1, \ldots, c_n)^T$ Spaltenvektor.
- $\mathbf{x} = (x_1, \ldots, x_n)^T$ Spaltenvektor.
- $\mathbf{0} = (0, \ldots, 0)^T$ Vektor aus Nullen.

Beispiel

Das LP

$$\begin{aligned} \text{Max.} \quad & 13x_1 + 10x_2 + 6x_3 \\ \text{s.d.} \quad & 5x_1 + x_2 + 3x_3 \leq 8 \\ & 3x_1 + x_2 \leq 3 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

in Matrix-Vektor Notation

$$\text{Max.} \begin{bmatrix} 13 & 10 & 6 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\text{s.d.} \begin{bmatrix} 5 & 1 & 3 \\ 3 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 8 \\ 3 \end{bmatrix}$$

$$\mathbf{x} \geq \mathbf{0}$$

Konstruktion duales LP

- Sei y_i der Faktor für Nebenbed. i und $\mathbf{y}$ Spaltenvektor der y_i's.
- Da $x_j \geq 0$, suchen wir eine obere Schranke für c_j:

$$c_j \leq \sum_{i=1}^{n} y_i a_{i,j} = \mathbf{y}^T \mathbf{a}_{\cdot,j}$$

  wobei $\mathbf{a}_{\cdot,j}$ der j-te Spaltenvektor von $\mathbf{A}$.
- Insgesamt also

$$\mathbf{c}^T \leq \mathbf{y}^T \mathbf{A}$$

- Wir verlangen $\mathbf{y} \geq \mathbf{0}$, und mit $\mathbf{x} \geq \mathbf{0}$ ergibt sich

$$\mathbf{c}^T \mathbf{x} \leq (\mathbf{y}^T \mathbf{A})\mathbf{x} = \mathbf{y}^T (\mathbf{A}\mathbf{x}) \leq \mathbf{y}^T \mathbf{b},$$

  da $\mathbf{x}$ eine Lösung für das primale LP ist.
- Unser Ziel die Minimierung von $\mathbf{y}^T \mathbf{b}$.

Kochrezept

$$\begin{array}{lrcl} \text{Max.} & \mathbf{c}^T\mathbf{x} & & \\ \text{s.d.} & \mathbf{Ax} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array} \qquad \begin{array}{lrcl} \text{Min.} & \mathbf{y}^T\mathbf{b} & & \\ \text{s.d.} & \mathbf{y}^T\mathbf{A} & \geq & \mathbf{c}^T \\ & \mathbf{y} & \geq & \mathbf{0} \end{array}$$

- $\mathbf{c}^T$ primale Zielfunktion $\longleftrightarrow$ $\mathbf{c}^T$ duale rechte Seite
- $\mathbf{b}$ primale rechte Seite $\longleftrightarrow$ $\mathbf{b}$ duale Zielfunktion
- Primale Nebenbedingungen über Zeilen der Matrix $\mathbf{A}$ $\longleftrightarrow$ Duale Nebenbedingungen über Spalten der Matrix $\mathbf{A}$
- Maximierung und $\leq$-Bedingungen im primalen LP $\longleftrightarrow$ Minimierung und $\geq$-Bedingungen im dualen LP

Beispiel

Primal:

$$\begin{aligned} \text{Max.} \quad & 13x_1 + 10x_2 + 6x_3 \\ \text{s.d.} \quad & 5x_1 + x_2 + 3x_3 \leq 8 \\ & 3x_1 + x_2 \leq 3 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Dual:

$$\begin{aligned} \text{Min.} \quad & 8y_1 + 3y_2 \\ \text{s.d.} \quad & 5y_1 + 3y_2 \geq 13 \\ & y_1 + y_2 \geq 10 \\ & 3y_1 \geq 6 \\ & y_1, y_2 \geq 0 \end{aligned}$$

Beispiel

Primal:

$$\text{Max.} \begin{bmatrix} 13 & 10 & 6 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\text{s.d.} \begin{bmatrix} 5 & 1 & 3 \\ 3 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leq \begin{bmatrix} 8 \\ 3 \end{bmatrix}$$

$$\mathbf{x} \geq \mathbf{0}$$

Dual:

$$\text{Min.} \begin{bmatrix} y_1 & y_2 \end{bmatrix} \cdot \begin{bmatrix} 8 \\ 3 \end{bmatrix}$$

$$\text{s.d.} \begin{bmatrix} y_1 & y_2 \end{bmatrix} \cdot \begin{bmatrix} 5 & 1 & 3 \\ 3 & 1 & 0 \end{bmatrix} \geq \begin{bmatrix} 13 & 10 & 6 \end{bmatrix}$$

$$\mathbf{y} \geq \mathbf{0}$$

Schwache Dualität

Seien $\mathbf{x}$ und $\mathbf{y}$ jeweils zulässige Lösungen des primalen (Max-) und dualen (Min-)LPs. Dann gilt:

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{y}^T\mathbf{b} \ .$$

Jede primale Lösung erzielt höchstens so viel Wert wie jede duale Lösung.

Beweis:

$$\mathbf{c}^T\mathbf{x} \underset{(a)}{\leq} (\mathbf{y}^T\mathbf{A})\mathbf{x} = \mathbf{y}^T(\mathbf{A}\mathbf{x}) \underset{(b)}{\leq} \mathbf{y}^T\mathbf{b}$$

(a) da $\mathbf{x} \geq \mathbf{0}$ und $\mathbf{y}$ die dualen Nebenbedingungen erfüllt.

(b) da $\mathbf{y} \geq \mathbf{0}$ und $\mathbf{x}$ die primalen Nebenbedingungen erfüllt.

□

Dualität

Starke Dualität

Seien **x** und **y** jeweils optimale Lösungen des primalen (Max-) und dualen (Min-)LPs. Dann gilt:

$$\mathbf{c}^T\mathbf{x} = \mathbf{y}^T\mathbf{b} \ .$$

Alle optimalen Lösungen der LPs erzielen den gleichen Wert.

- **Das zentrale Resultat** im Bereich der linearen Optimierung!
- Strukturbeziehungen zwischen Optimierungsproblemen
- Viele wichtige Konsequenzen für Approx.algorithmen (Vorlesung, evtl. WS 23/24)

Beispiele

Primales LP für (fraktionales) maximum Matching im Graph G:

$$\begin{aligned} \text{Max.} \quad & \sum_{e \in E} x_e & & & \\ \text{s.d.} \quad & \sum_{e=\{u,v\}} x_e & \leq & 1 & \text{für alle } v \in V \\ & x_e & \geq & 0 & \text{für alle } e \in E \end{aligned}$$

Das duale LP

$$\begin{aligned} \text{Min.} \quad & \sum_{v \in V} y_v & & & \\ \text{s.d.} \quad & y_u + y_v & \geq & 1 & \text{für alle } \{u, v\} \in E \\ & y_v & \geq & 0 & \text{für alle } v \in V \end{aligned}$$

ist das LP für (fraktionales) Vertex Cover!

Konsequenzen

- Jedes Matching M ist eine integrale Lösung $\mathbf{x}$ für das primale LP.
- Jede Überdeckung $\ddot{U}$ ist eine integrale Lösung $\mathbf{y}$ für das duale LP.
- Schwache Dualität: $|M| \leq |\ddot{U}|$
- Wenn es Matching M und Überdeckung $\ddot{U}$ gibt mit $|M| = |\ddot{U}|$, dann sind beide optimal!
- Starke Dualität:
  Wert des besten fraktionalen Matching
  = Wert des kleinsten fraktionalen Vertex Cover

Zusammenfassung

- Flüsse sind ein grundlegendes Modell der Optimierung. Maximale Flüsse können effizient berechnet werden.
  - Maximale Flüsse und minimale Schnitte sind eng verwandt (genauer gesagt: dual zueinander).
  - Der Ford-Fulkerson Algorithmus zeigt, dass für ganzzahlige Kapazitäten immer ein ganzzahliger Maximalfluss existiert.
  - Der Edmonds-Karp Algorithmus berechnet maximale Flüsse in (stark) polynomieller Zeit.
- Maximale Flüsse sind sehr hilfreich für andere Probleme, z.B. zur effizienten Optimierung in bipartiten Graphen:
  - Maximum Matchings
  - Minimum Vertex Cover
  - Maximum Matching mit minimalen Kosten

Zusammenfassung

- Lineare Programmierung ist eine mächtige Technik aus der kontinuierlichen Optimierung.
  - Lineares Programm (LP) hat eine lineare Zielfunktion und lineare Nebenbedingungen.
  - Es gibt effiziente Algorithmen zur Lösung von LPs.
  - Geometrie: Optimallösungen sind Ecken des Lösungpolytops
  - I.d.R. nur fraktionale Ecken. Ausnahme: LP für maximum Matching in bipartiten Graphen hat nur integrale Ecken.

- Dualität linearer Programme
  - LPs kommen immer in Paaren:
    Duales LP soll möglichst gute Schranke an den Optimalwert des primalen LPs liefern (und umgekehrt).
  - Schwache und starke Dualität:
    Wert jeder Lösung des primalen (Max-)LPs immer kleiner als Wert jeder Lösung des dualen (Min-)LPs. Optimalwerte sind gleich.