

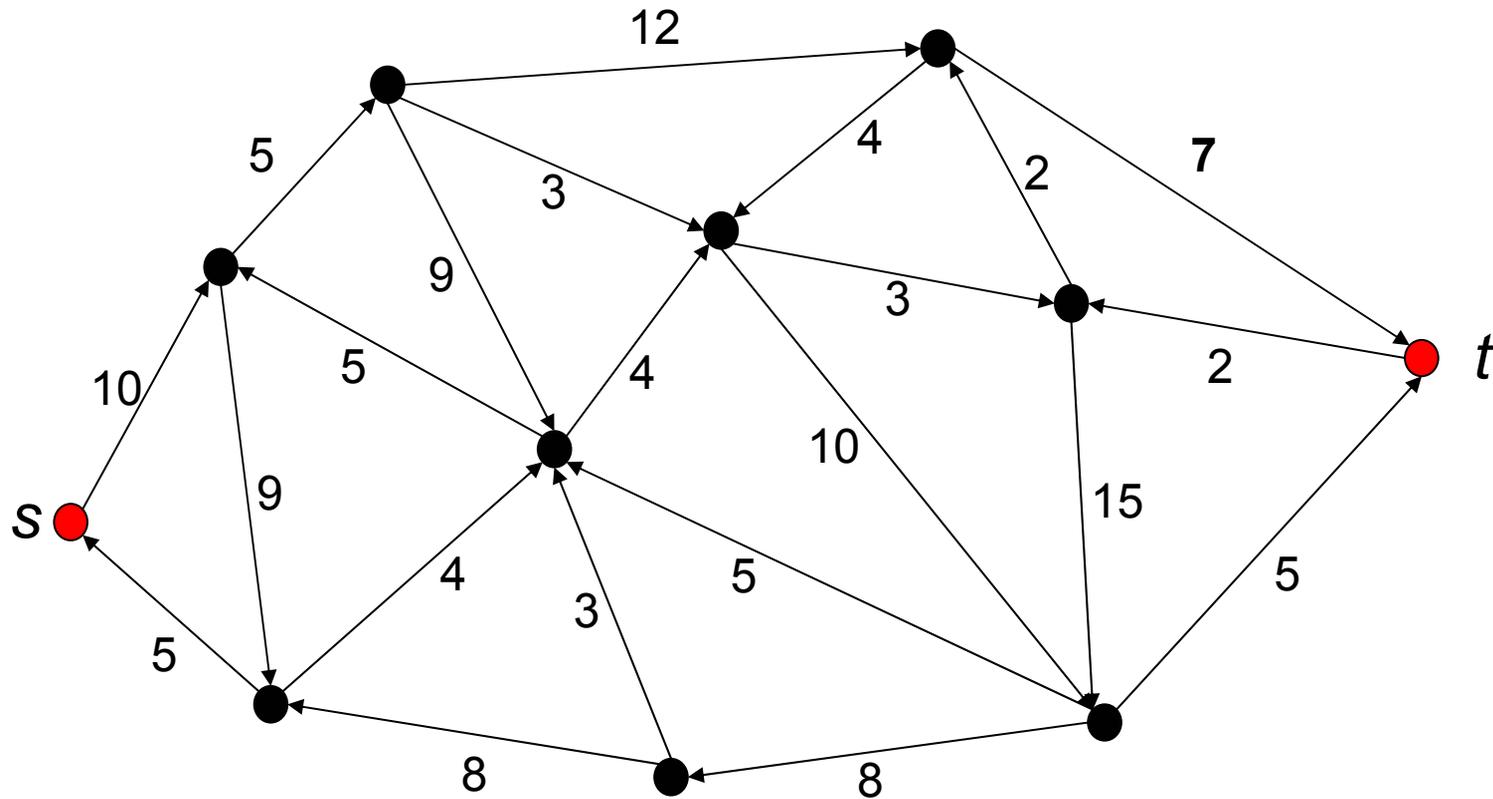


Algorithmentheorie

17 - Maximale Flüsse

Prof. Dr. S. Albers

1. Maximale Flüsse in Netzwerken



Netzwerke und Flüsse

$N = (V, E, c)$ gerichtetes Netzwerk

$G = (V, E)$ gerichteter Graph, $c: E \rightarrow \mathbb{R}^+$ Kapazitätsfunktion

$s, t \in V$, s Quelle, t Senke

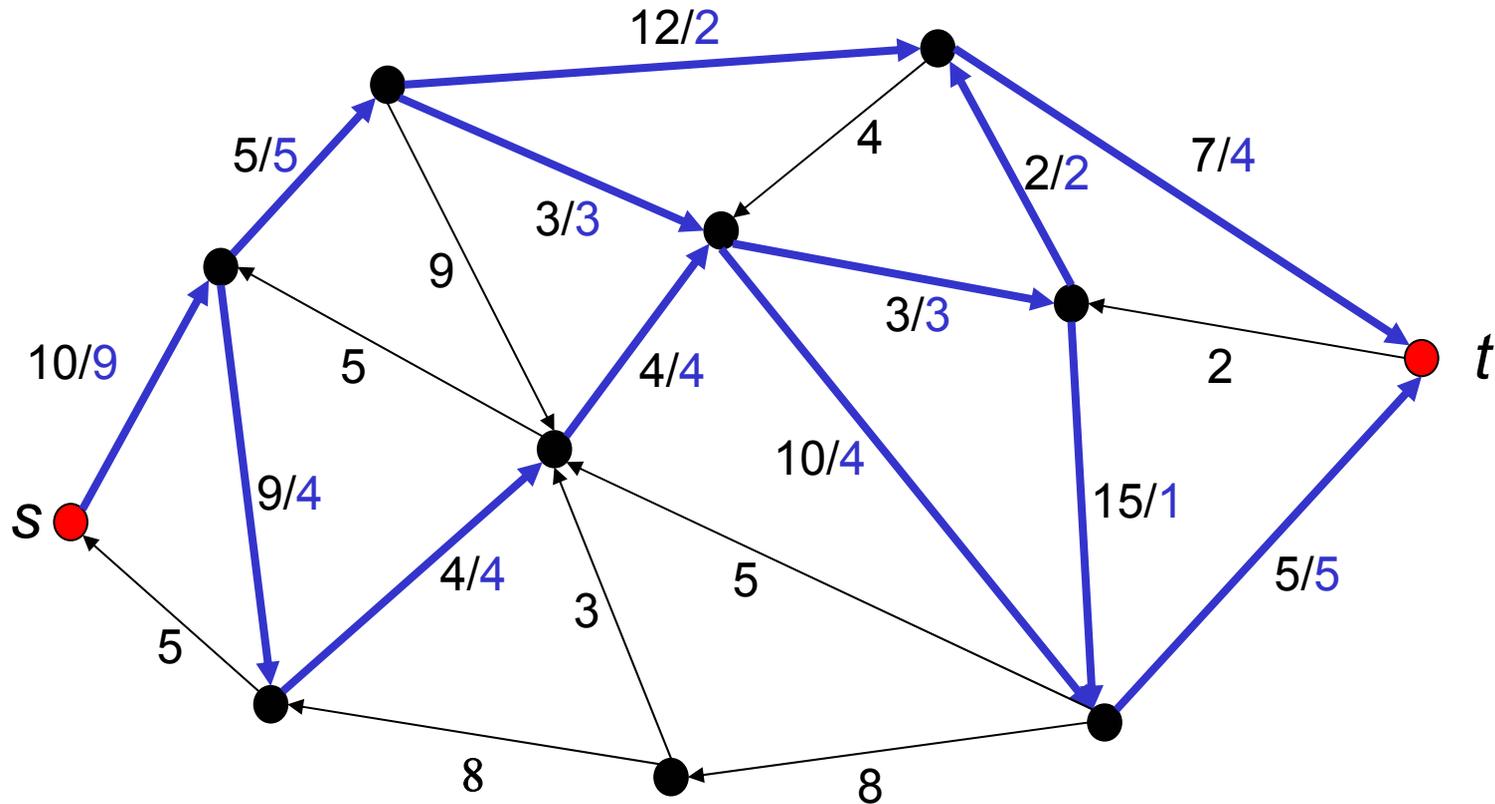
Zulässiger (s, t) -Fluss: $f: E \rightarrow \mathbb{R}$

a) $0 \leq f(e) \leq c(e) \quad \forall e \in E$ Kapazitätsbeschränkung

b) $\sum_{e \in \text{ein}(v)} f(e) = \sum_{e \in \text{aus}(v)} f(e) \quad \forall v \in V - \{s, t\}$ Flusserhaltung

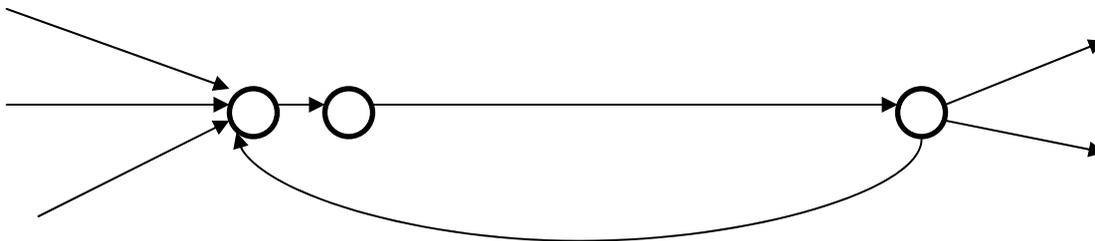
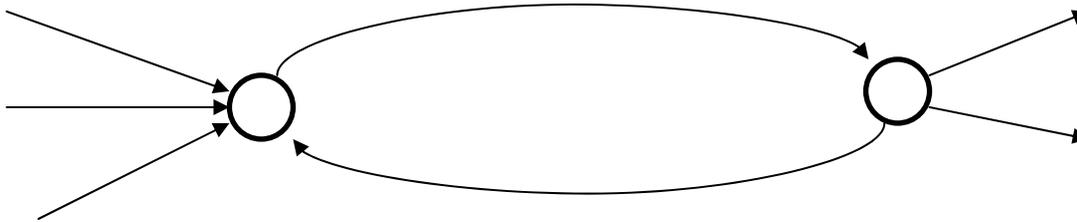
$\text{ein}(v) = \{\text{Kanten in } v \text{ hinein}\} \quad \text{aus}(v) = \{\text{Kanten aus } v \text{ heraus}\}$

Beispiel



Doppelkanten

O.B.d.A. hat der Graph keine Doppelkanten.



Der Wert eines Flusses

Sei f ein zulässiger Fluss Dann ist sein **Wert**:

$$W(f) = \sum_{e \in \text{aus}(s)} f(e) - \sum_{e \in \text{ein}(s)} f(e)$$

Das Max-Fluss Problem:

Berechne einen zulässigen Fluss maximalen Werts.

2. Schnitte

Definition: Ein (s,t) -Schnitt ist eine Partition S, T von V , d.h. $V = S \cup T$, $S \cap T = \emptyset$, so dass $s \in S, t \in T$.

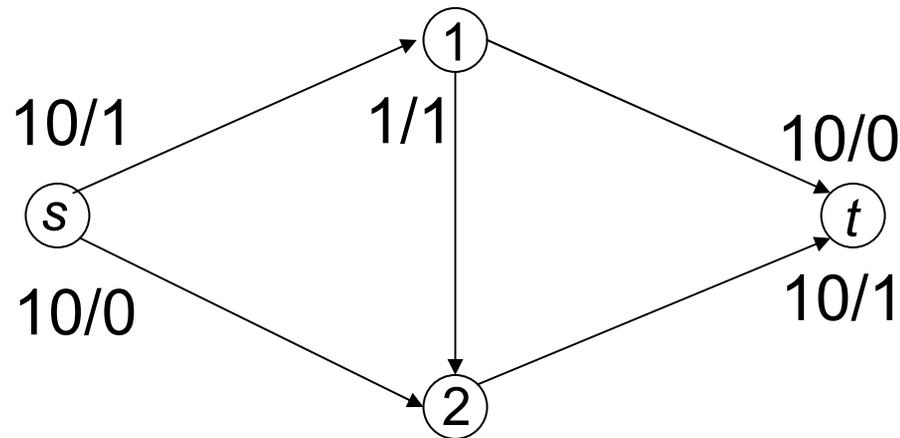
Kapazität eines Schnitts:
$$C(S, T) = \sum_{e \in E \cap (S \times T)} c(e)$$

Satz 1: Seien f ein maximaler Fluss und (S, T) ein minimaler (s,t) -Schnitt. Dann gilt:

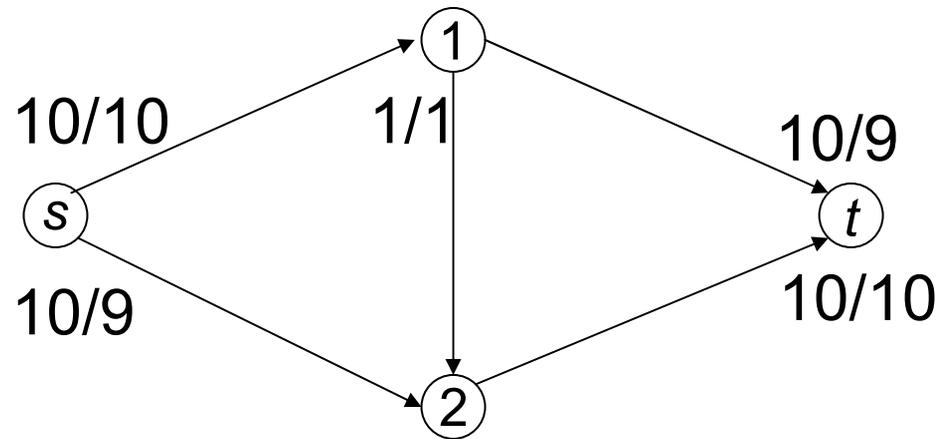
$$W(f) = C(S, T)$$

3. Algorithmische Idee

Zunehmende Wege: Finde Wege, entlang deren der Fluss erhöht werden kann.



Zunehmende Wege



4. Restnetzwerke

Restnetzwerk RN bzgl. eines zulässigen Flusses f :

$$E_1 = \{ (v,w) : (v,w) = e \in E \text{ und } f(e) < c(e) \}$$

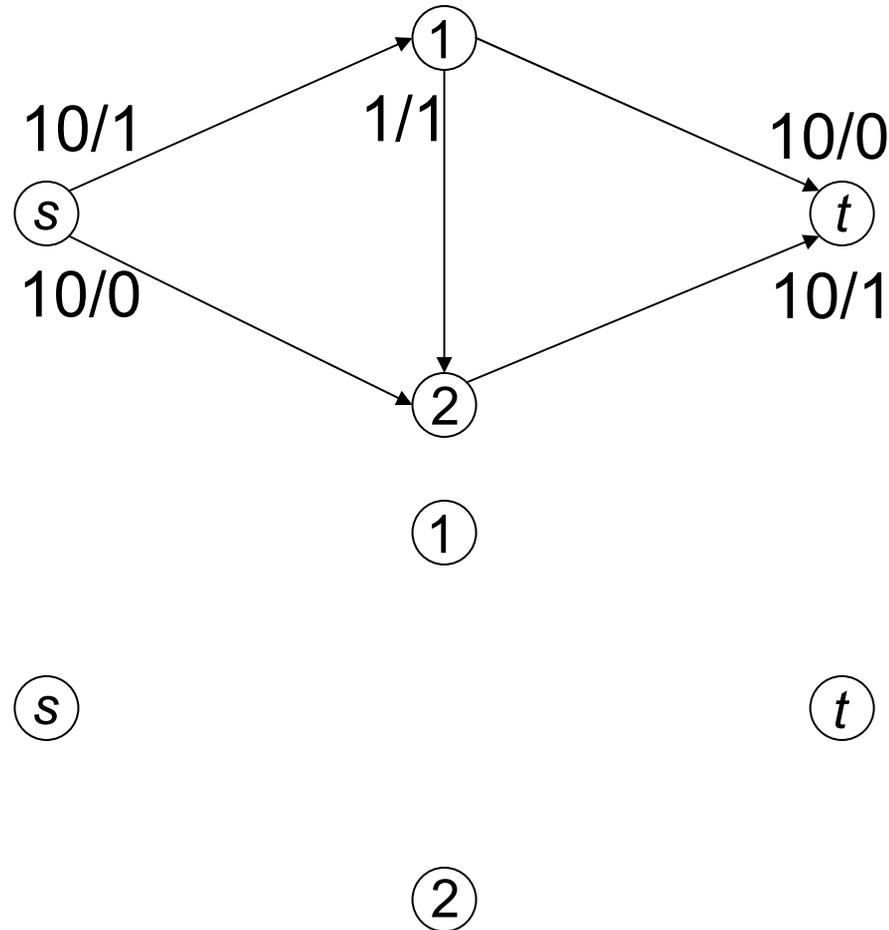
$$E_2 = \{ (w,v) : (v,w) = e \in E \text{ und } f(e) > 0 \}$$

Für $e = (v,w) \in E$ verwende e_1 für $(v,w) \in E_1$ (sofern existent)
 e_2 für $(w,v) \in E_2$ (sofern existent)

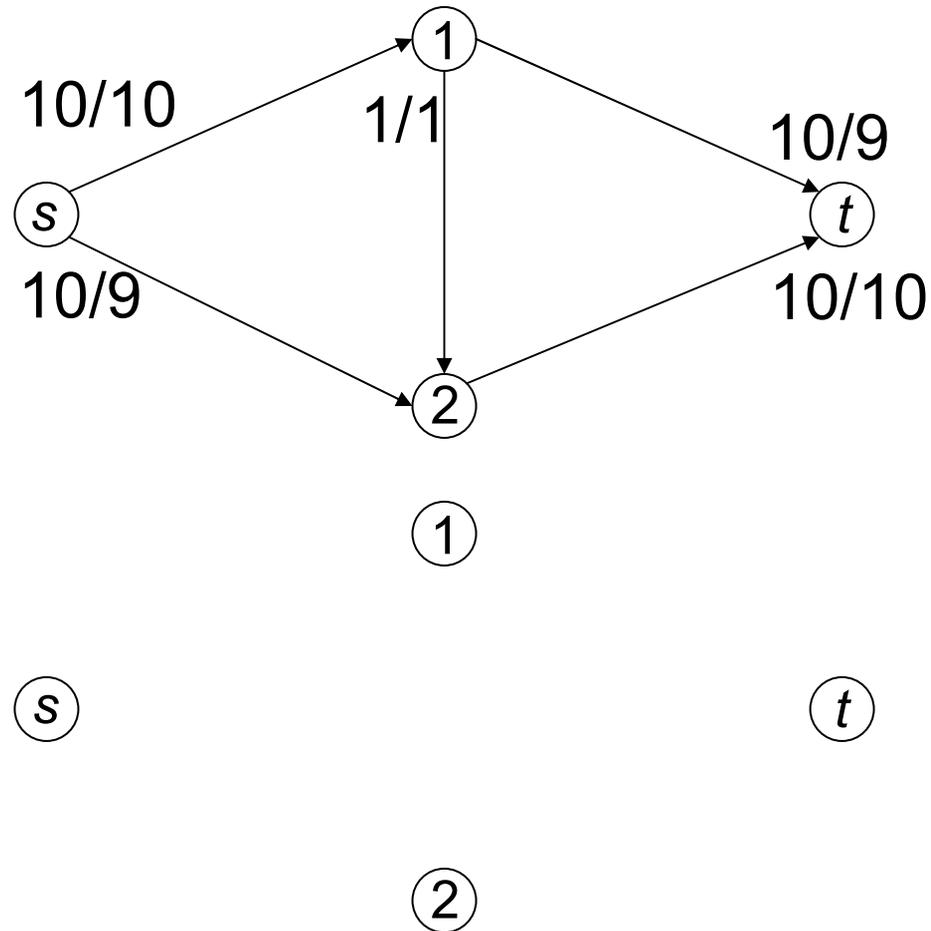
$$\bar{c} : E_1 \cup E_2 \rightarrow \mathbb{R}^+ \quad \begin{array}{ll} \bar{c}(e_1) = c(e) - f(e) & \text{für } e_1 \in E_1 \\ \bar{c}(e_2) = f(e) & \text{für } e_2 \in E_2 \end{array}$$

$$RN = (V, E_1 \cup E_2, \bar{c})$$

Beispiel



Beispiel



Schichtnetzwerke

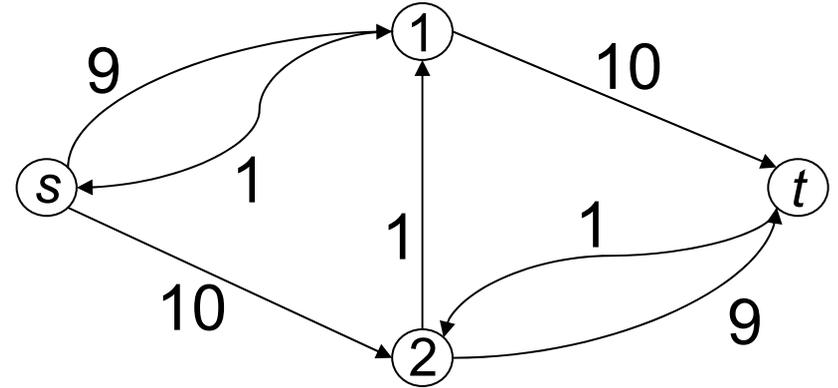
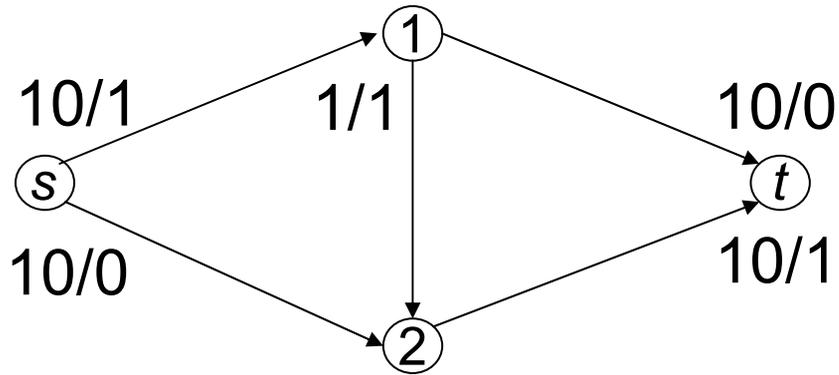
$$V_0 = \{s\}$$

$$V_{i+1} = \{w \in V - (V_0 \cup \dots \cup V_i); \exists v \in V_i : (v, w) \in E_1 \cup E_2\} \quad \text{für } i \geq 1$$

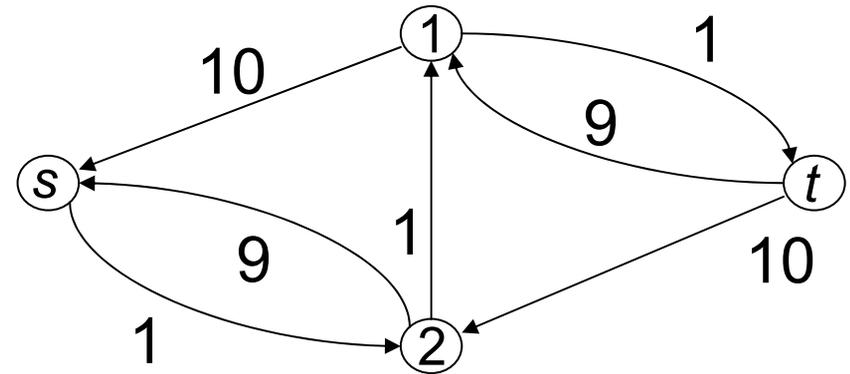
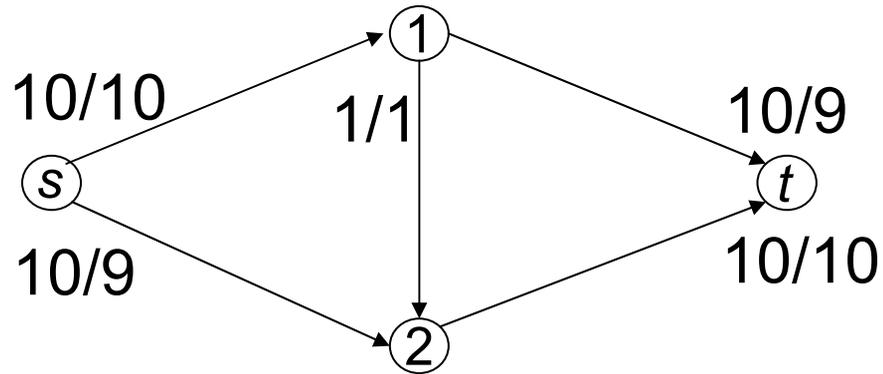
$$\bar{V} = \bigcup_{i \geq 0} V_i$$

$$SN = \left(\bar{V}, (E_1 \cup E_2) \cap \bigcup_{i \geq 0} (V_i \times V_{i+1}), \bar{c} \right)$$

Beispiel



Beispiel



Maximale Flüsse

Lemma 1: Sei f ein zulässiger (s,t) -Fluss in N und sei $SN = (\bar{V}, \bar{E}, \bar{c})$ das Schichtnetzwerk bzgl. f .

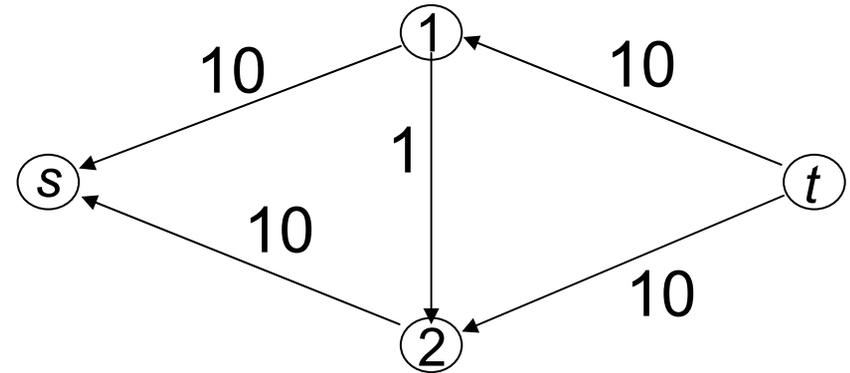
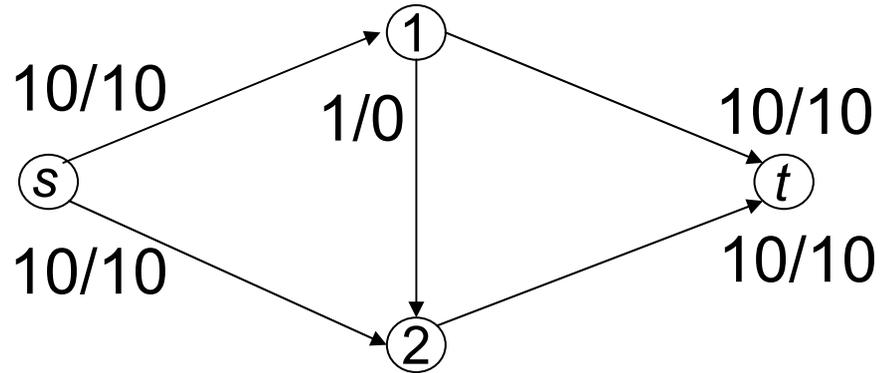
- a) f ist ein maximaler Fluss gdw. $t \notin \bar{V}$
b) Sei \bar{f} ein zulässiger (s,t) -Fluss in SN . Dann ist $f' : E \rightarrow R$ mit

$$f'(e) = f(e) + \bar{f}(e_1) - \bar{f}(e_2)$$

ein zulässiger (s,t) -Fluss in N mit $W(f') = W(f) + W(\bar{f})$

Definiere $\bar{f}(e_i) = 0$ für $e_i \notin \bar{E}$.

Beispiel



Beweis, Teil b)

Beweis: b) Kapazitätsbedingungen. Sei $e \in E$, dann gilt:

$$0 \leq f(e) - \bar{f}(e_2)$$

$$\leq f'(e)$$

$$\leq f(e) + \bar{f}(e_1)$$

$$\leq c(e)$$

Beweis, Teil b)

Für jeden Knoten $v \in V$ gilt:

$$\begin{aligned} & \sum_{e \in \text{aus}(v)} f'(e) - \sum_{e \in \text{ein}(v)} f'(e) \\ &= \sum_{e \in \text{aus}(v)} f(e) - \sum_{e \in \text{ein}(v)} f(e) + \left(\sum_{e \in \text{aus}(v)} \bar{f}(e_1) + \sum_{e \in \text{ein}(v)} \bar{f}(e_2) \right) \\ & \quad - \left(\sum_{e \in \text{ein}(v)} \bar{f}(e_1) + \sum_{e \in \text{aus}(v)} \bar{f}(e_2) \right) \end{aligned}$$

Flusserhaltung: Für $v \in V - \{s, t\}$ ist der Ausdruck 0

Wert: Für $v = s$ ist $W(f') = W(f) + W(\bar{f})$

Beweis, Teil a)

a) " \Rightarrow "

Sei $t \in \bar{V}$.

Dann existiert ein Weg P von s nach t in SN .



$\varepsilon = \min.$ Kapazität von Kanten in P

$$\bar{f}(e) = \begin{cases} \varepsilon & e \text{ auf } P \\ 0 & e \text{ nicht auf } P \end{cases}$$

Also ist f nicht maximal.

Beweis, Teil a)

" \Leftarrow "

Setze $S = \bar{V}$, $T = V - S$

Dann gilt $s \in S$, $t \in T$, und (S, T) ist ein (s, t) -Schnitt.

$$(E_1 \cup E_2) \cap (S \times T) = \emptyset$$

$$f(e) = c(e) \quad \text{für } e \in S \times T$$

$$f(e) = 0 \quad \text{für } e \in T \times S$$

$$W(f) = \sum_{e \in E \cap (S \times T)} f(e) - \sum_{e \in E \cap (T \times S)} f(e) = C(S, T)$$

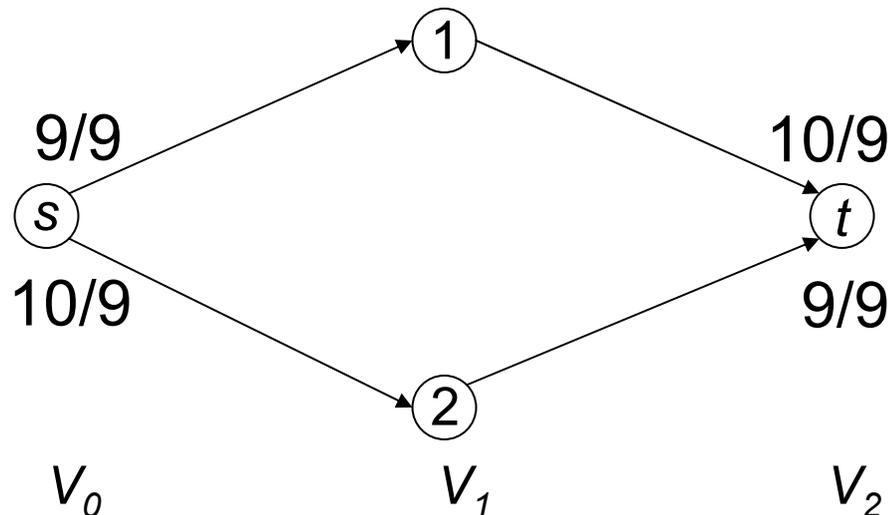
Da $W(g) \leq C(S, T)$ für jeden zulässigen Fluss g , ist **f ein maximaler Fluss.**

5. Blockierende Flüsse

Definition: Ein zulässiger Fluss \bar{f} in einem Schichtnetzwerk SN ist **blockierend**, wenn auf jedem Weg

$$s = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \xrightarrow{e_3} \dots \xrightarrow{e_k} v_k = t$$

von s nach t mindestens eine Kante **gesättigt** ist, d.h. $\bar{f}(e_i) = \bar{c}(e_i)$ für mindestens ein i .



Algorithmus

1. $f(e) \leftarrow 0$ für alle $e \in E$;
2. Konstruiere Schichtnetzwerk $SN = (\bar{V}, \bar{E}, \bar{c})$ bzgl. f ;
3. **while** $t \in \bar{V}$ **do**
4. Finde einen blockierenden Fluss \bar{f} in SN ;
5. Aktualisiere f gemäß \bar{f} wie in Lemma 2b) beschrieben;
6. Konstruiere Schichtnetzwerk SN bzgl. f ;
7. **endwhile**;

Wie findet man einen blockierenden Fluss?

Wie viele Iterationen?

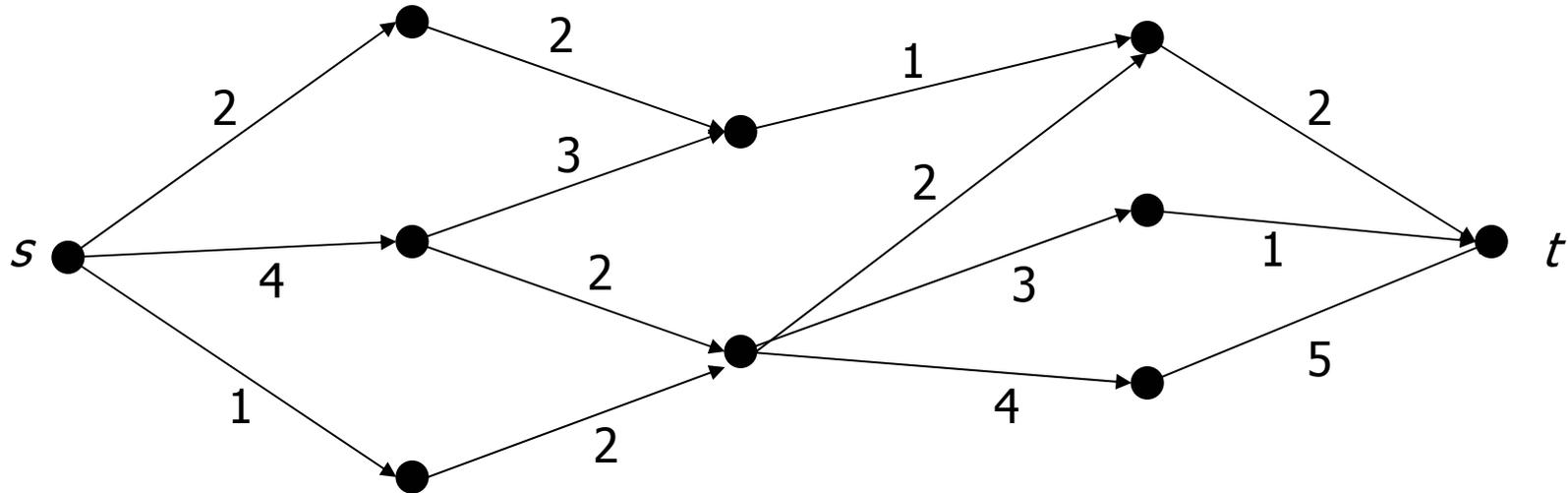
6. Die Tiefe eines Schichtnetzwerks

Definition: Die Tiefe eines Schichtnetzwerks SN ist das k mit $t \in V_k$.

Lemma 2: Sei k_i die Tiefe des Schichtnetzwerks in der i -ten Iteration.
Dann gilt $k_i > k_{i-1}$, für $i \geq 2$.

Korollar: Die Anzahl der Iterationen ist $\leq n$.

7. Blockierende Flüsse: DFS-Algorithmus



Beginne bei s und wähle stets die **erste ausgehende Kante** aus einem Knoten bis **a) t erreicht** oder **b) Sackgasse v erreicht**.

(a) Bestimme die kleinste Kap. ε entlang d. Wegs. Erhöhe den Fluss um ε , vermindere die Kap. um ε und entferne gesättigte Kanten.

(b) Gehe einen Knoten zurück, entferne v und seine eingehenden Kanten.

Satz 2: Ein blockierender Fluss kann in Zeit $O(ne)$ berechnet werden.

Beweis: k =Tiefe des Schichtnetzwerks

Konstruktion eines Wegs benötigt Zeit

$O(k + \# \text{ durchlaufene Kanten, die in Sackgassen enden})$.

Höchstens e Wege werden konstruiert.

Gesamtzeit: $O(ke + e) = O(ne)$

Verbesserter Algorithmus

Arbeite mit dem Schichtnetzwerk.

Potenzial eines Knotens v bzgl. f

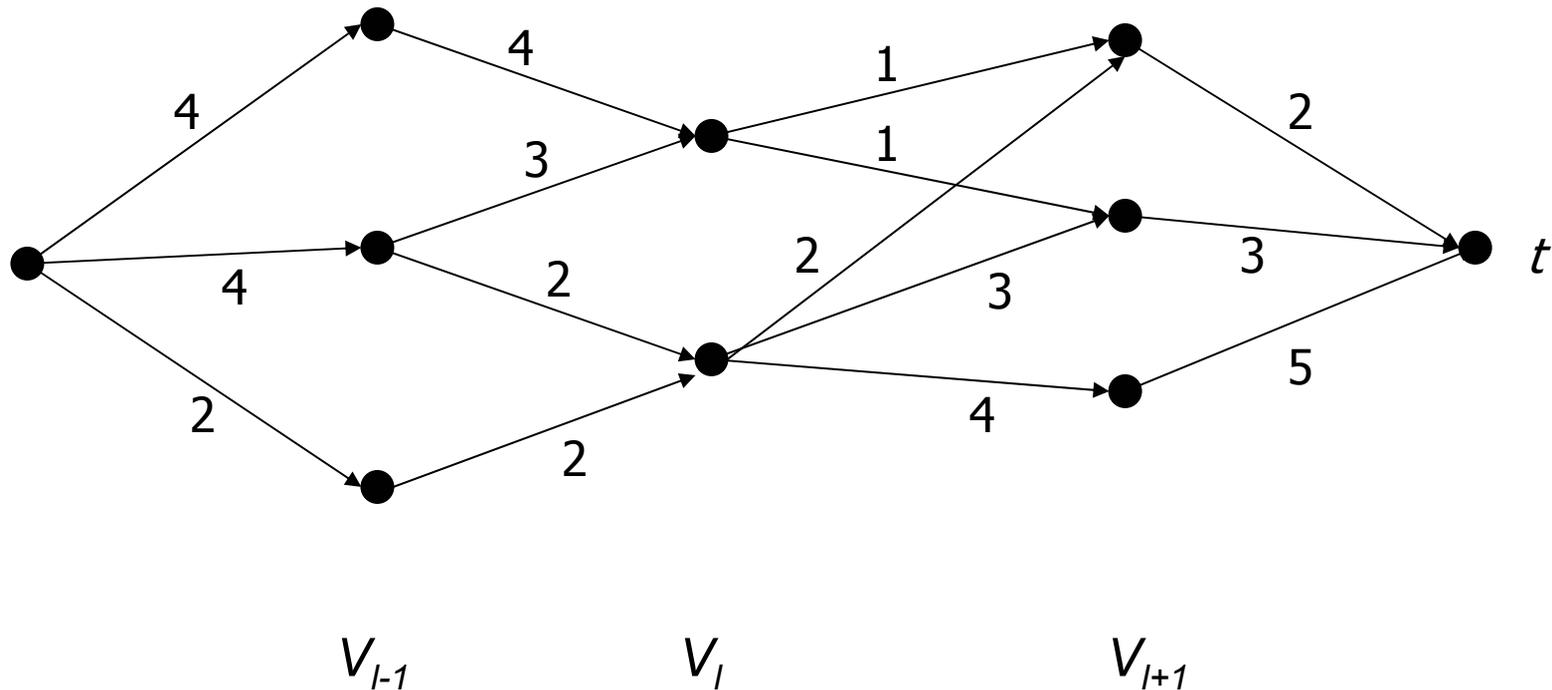
$$PO(v) = \min \left\{ \sum_{e \in \text{aus}(v)} c(e) - f(e), \sum_{e \in \text{ein}(v)} c(e) - f(e) \right\}$$

$$PO^* = \min \{PO(v) : v \in V\}$$

Verbesserter Algorithmus

Wähle v mit $PO(v) = PO^*$.

Schiebe PO^* Flusseinheiten von v in höhere Schichten.



Verbesserter Algorithmus

Schicht V_h : Menge $S_h \subseteq V_h$, die PO^* zusätzliche Flusseinheiten hat.

$$PO^* = \sum_{x \in S_h} S[x], \quad S[x] = \text{Überfluss am Knoten } x.$$

Ziehe PO^* Flusseinheiten nach v aus niedrigeren Schichten.

Fluss erhöht sich um PO^* Einheiten.

Vereinfache das Netzwerk, indem gesättigte Kanten und Knoten mit Ein- oder Ausgangsgrad gleich 0 entfernt werden. (Mindestens ein Knoten wird entfernt.)

Das Schieben von Fluss

Algorithmus *schiebe*(x, S, h);

$\ll x$ ist Knoten in Schicht V_h und bei x sind S zusätzliche Flusseinheiten verfügbar. Diese werden in Knoten der Schicht V_{h+1} geschoben.

1. **while** $S > 0$ **do**
2. Sei $e = (x, y)$ die erste aus x ausgehende Kante;
3. $\delta \leftarrow \min(S, c(e) - f(e))$;
4. Erhöhe den Fluss auf e um δ , füge y zu S_{h+1} hinzu
 (falls noch nicht El.), erhöhe $S[y]$ um δ ;
5. $S \leftarrow S - \delta$;
6. **if** $c(e) = f(e)$ **then** entferne e aus dem Graphen **endif**;
7. **endwhile**;
8. Entferne x aus S_h und setze $S[x]$ auf Null;
9. **if** ($aus(x) = \emptyset$ und $x \neq t$) **or** ($ein(x) = \emptyset$ und $x \neq s$) **then**
10. Füge x zur Menge *del* hinzu;
11. **endif**;

Algorithmus für blockierenden Fluss

1. **for all** $x \in V$ **do** $S[x] \leftarrow 0$ **endfor**;
2. **for all** $l, 0 \leq l \leq k$, **do** $S_l \leftarrow \emptyset$ **endfor**;
3. $del \leftarrow \emptyset$
4. **while** SN ist nicht leer **do**
5. Berechne $PO[v]$ für alle $v \in V$ und $PO^* = \min \{PO[v]; v \in V\}$;
 Sei $v \in V_l$ ein Knoten mit $PO^* = PO[v]$;
6. $S[v] \leftarrow PO^*$; $S_l \leftarrow \{v\}$;
7. **for** h von l bis $k - 1$ **do**
8. **for all** $x \in S_h$ **do** schiebe($x, S[x], h$) **endfor**;
9. **endfor**;
10. $S[v] \leftarrow PO^*$; $S_l \leftarrow \{v\}$
11. **for** h von l bis 1 **do**
12. **for all** $x \in S_h$ **do** ziehe($x, S[x], h$) **endfor**;
13. **endfor**;
14. vereinfache(del)
15. **endwhile**;

Ergebnis

Satz 3: Ein blockierender Fluss in einem Schichtnetzwerk kann in Zeit $O(n^2)$ berechnet werden.

Beweis: 1-3: $O(n)$

Schleife 4-15: $O(n)$ -mal ausgeführt. Jede Ausführung kostet $O(n)$, wenn wir *schiebe*, *ziehe*, *vereinfache* ignorieren.

Alle Ausführungen von *schiebe* / *ziehe* benötigen Zeit $O(n + e)$.

Alle Ausführungen von *vereinfache* benötigen Zeit $O(n + e)$.

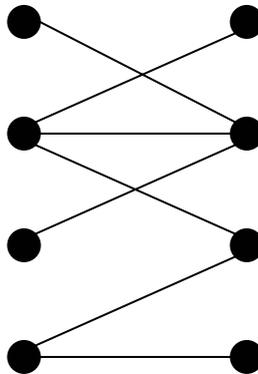
Ergebnis

Satz 4: Ein maximaler Fluss kann in Zeit $O(n^3)$ berechnet werden.

Beweis: Ein Schichtnetzwerk und ein blockierender Fluss können in Zeit $O(n^2)$ berechnet werden.

Matchings in bipartiten Graphen

Ein ungerichteter Graph $G = (V, E)$ ist **bipartit**, wenn $V = V_1 \cup V_2$ für $V_1, V_2 \subseteq V$ mit $V_1 \cap V_2 = \emptyset$ und $E \subseteq V_1 \times V_2$.



Matchings in bipartiten Graphen

Satz 5: Sei $G = (V_1 \cup V_2, E), E \subseteq V_1 \times V_2$, ein bipartiter Graph. Dann kann ein maximales Matching in Zeit $O(n^3)$ berechnet werden.

Beweis: Konstruiere Netzwerk wie folgt:
(Alle Kapazitäten sind 1.)

