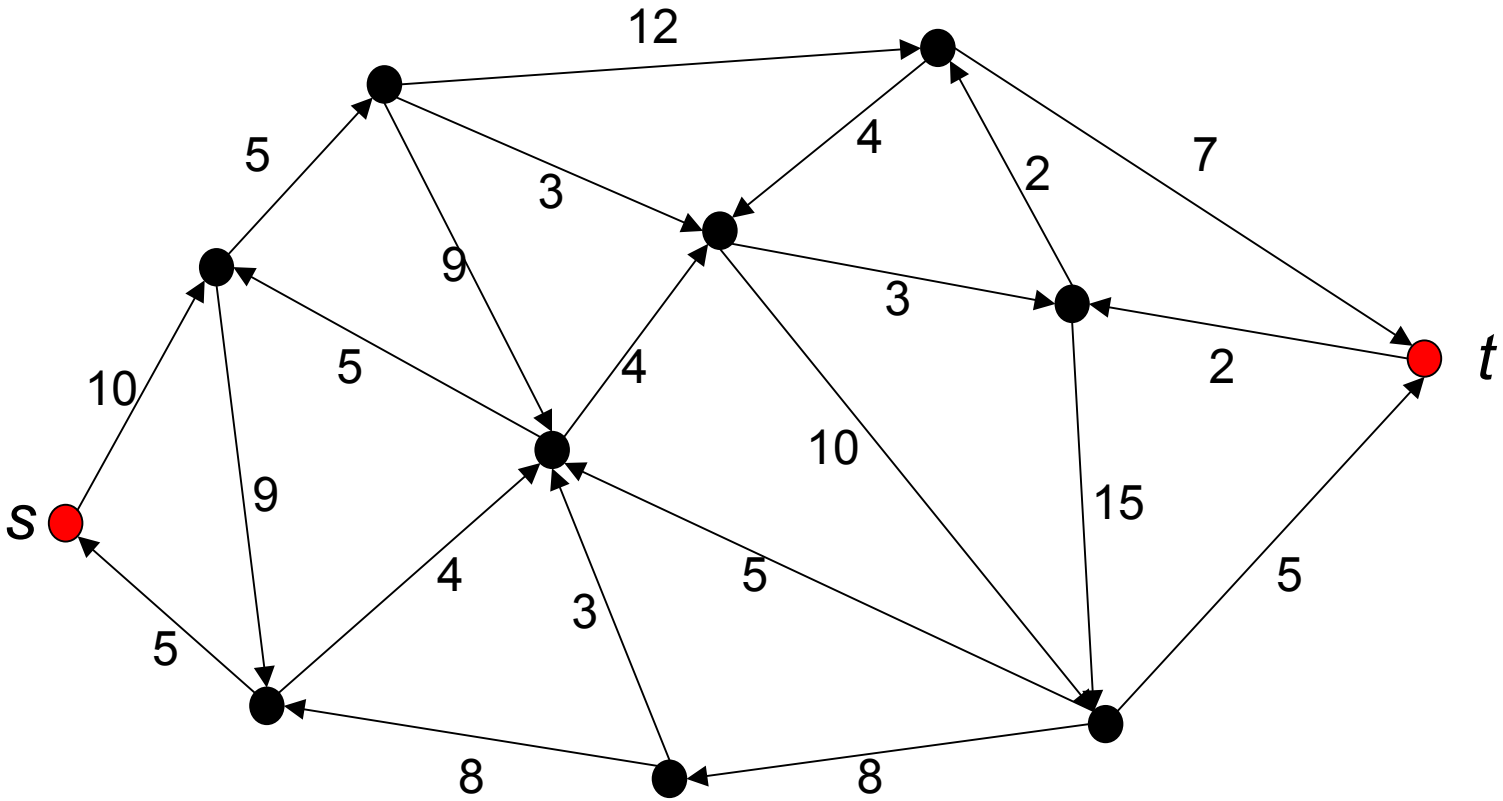
Algorithmentheorie

13 - Maximale Flüsse

Prof. Dr. S. Albers

Prof. Dr. Th. Ottmann

1. Maximale Flüsse in Netzwerken



Netzwerke und Flüsse

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

$G = (V,E)$ gerichteter Graph, $c: E \to R^+$ Kapazitätsfunktion

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

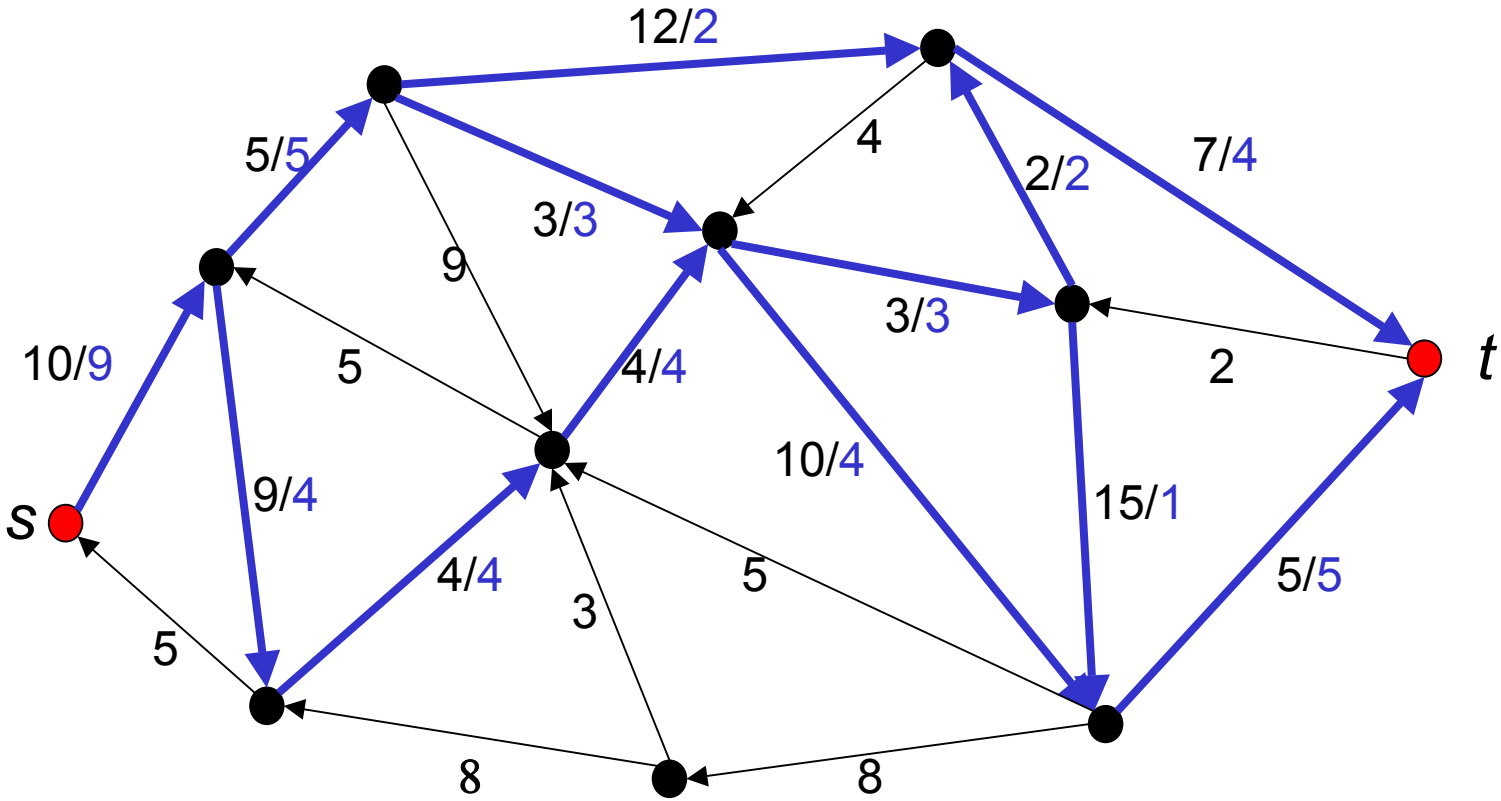
Zulässiger (s,t)-Fluss: $f: E \to R$

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

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

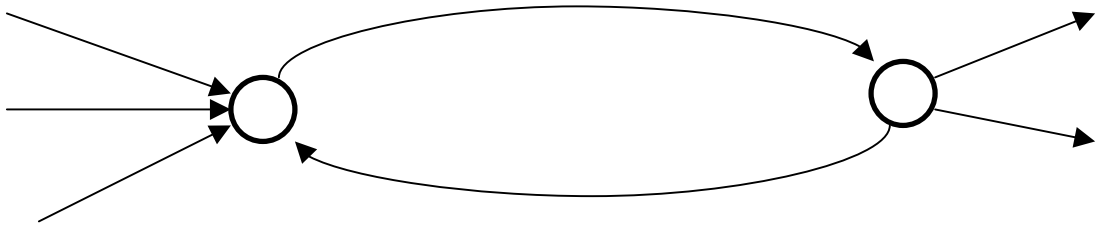
$ein(v) = \{\text{Kanten in } v \text{ hinein}\} \qquad aus(v) = \{\text{Kanten aus } v \text{ heraus}\}$

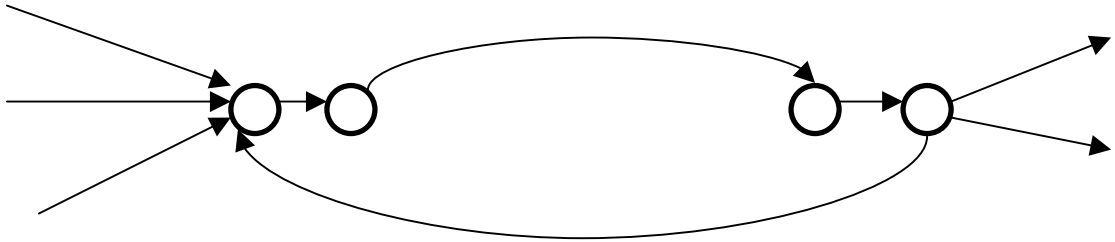
Beispiel



Doppelkanten

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





Der Wert eines Flusses

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

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

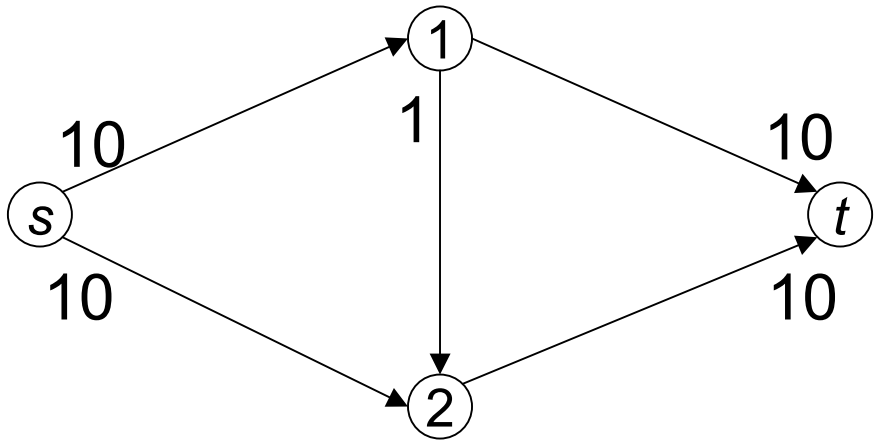
Das Max-Fluss Problem:

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



Flüsse und Schnitte

Lemma 1: Seien *f* ein zulässiger Fluss und (*S*,*T*) ein (*s*,*t*)-Schnitt.
Dann gilt:

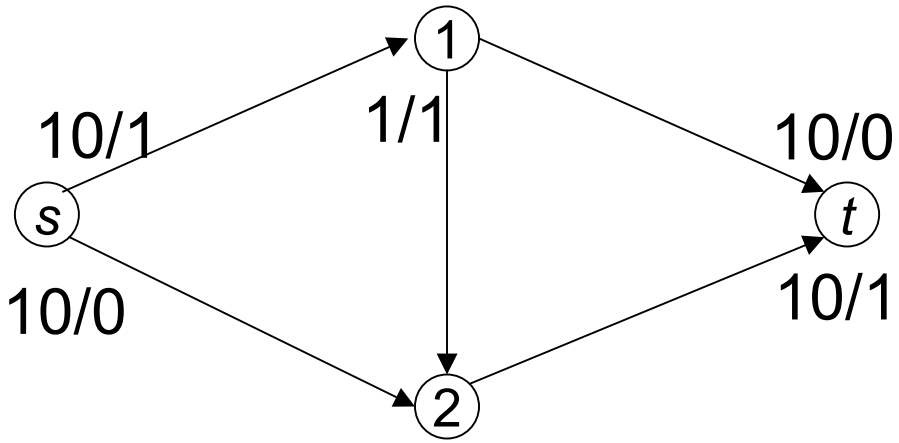
$$W(f) \leq C(S,T)$$

Beweis:

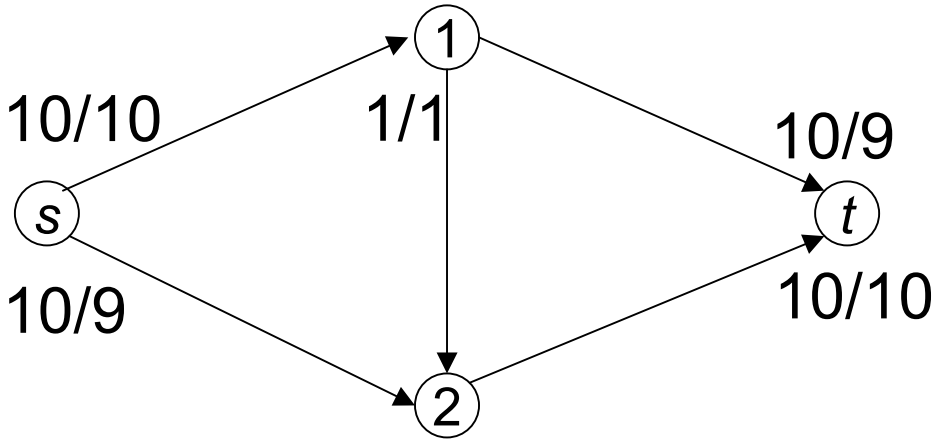
$$W(f) = \sum_{e \in aus(s)} f(e) - \sum_{e \in ein(s)} f(e)$$

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:*

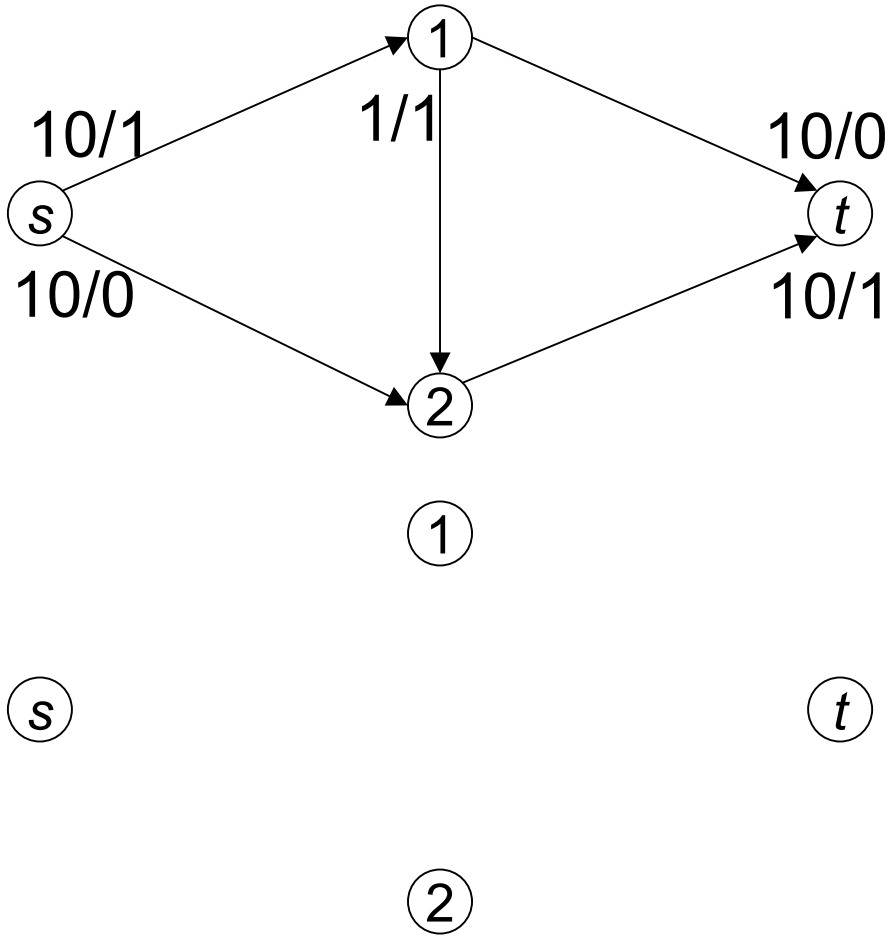
$$E_1 = \{ (v,w) : (v,w) = e \in E \text{ und } f(e) < c(e)\}$$
$$E_2 = \{ (w,v) : (v,w) = e \in E \text{ und } f(e) > 0\}$$

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)

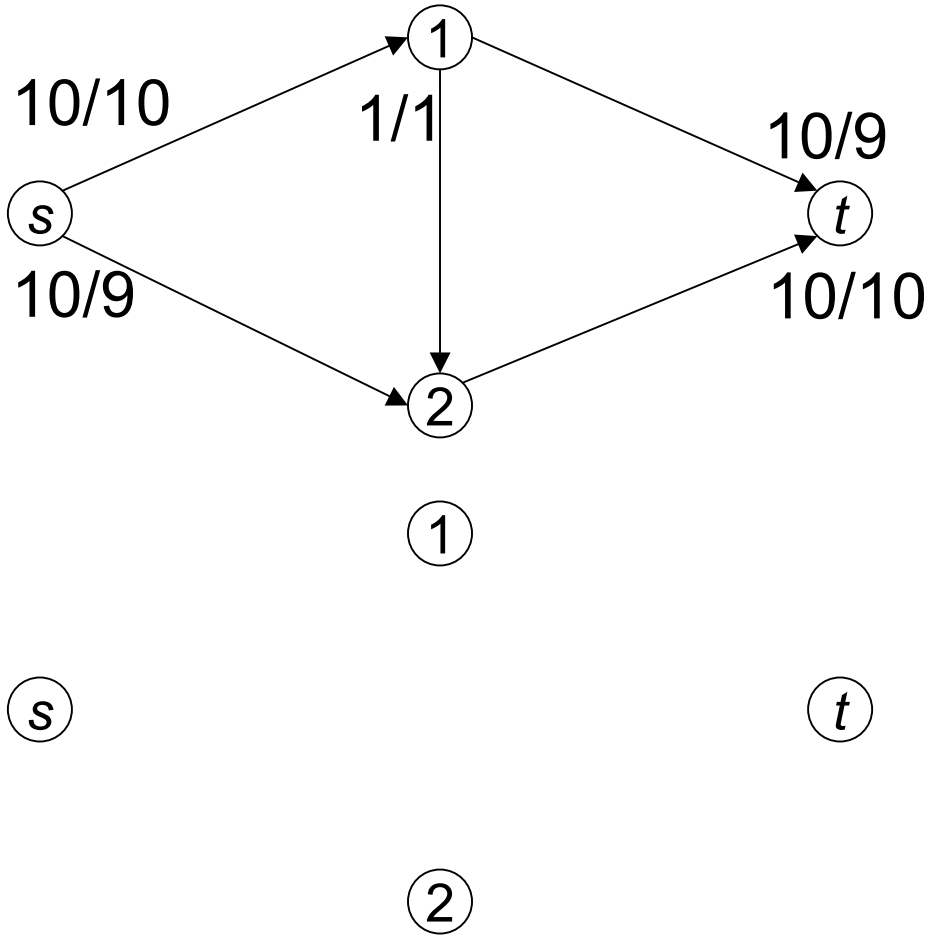
$$\bar{c} : E_1 \cup E_2 \rightarrow R^+ \qquad \bar{c}(e_1) = c(e) - f(e) \quad \text{für } e_1 \in E_1$$
$$\bar{c}(e_2) = f(e) \quad \text{für } e_2 \in E_2$$

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

Beispiel



Beispiel



Schichtnetzwerke

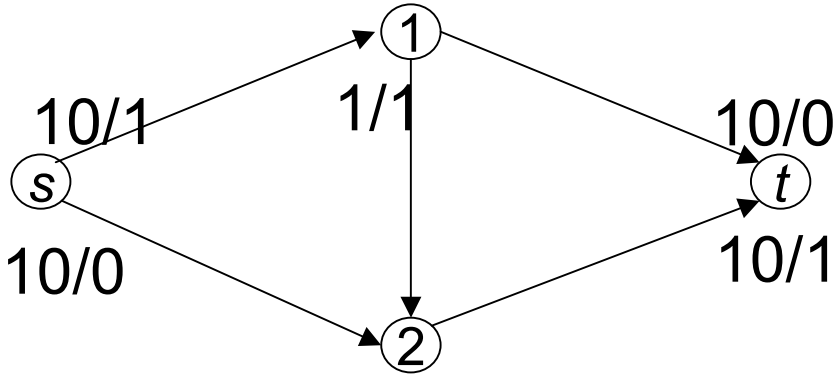
$$V_0 = \{s\}$$

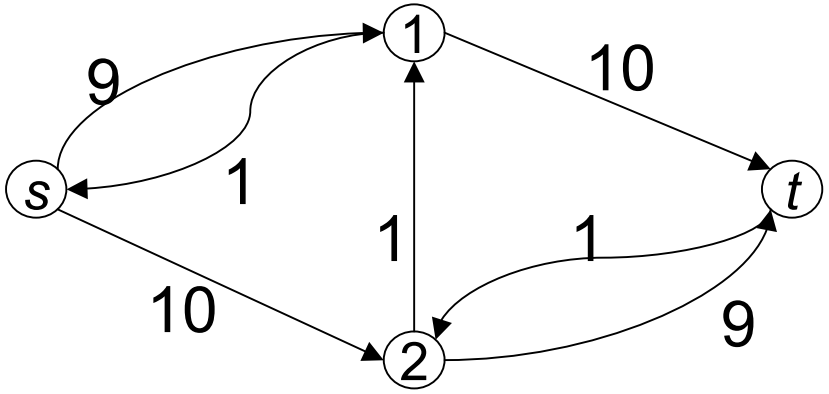
$$V_{i+1} = \{w \in V - (V_0 \cup \ldots \cup V_i);\ \ \exists v \in V_i : (v,w) \in E_1 \cup E_2\} \quad \text{für } i \geq 1$$

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

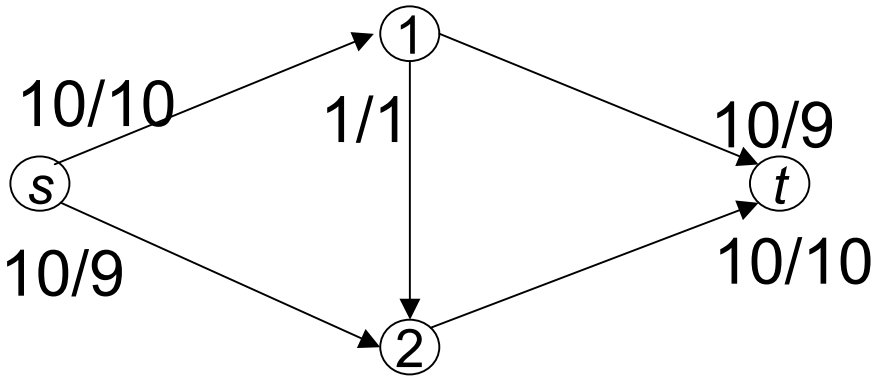
$$SN = \left(\overline{V}, (E_1 \cup E_2) \cap \bigcup_{i \geq 0} (V_i \times V_{i+1}), \overline{c} \right)$$

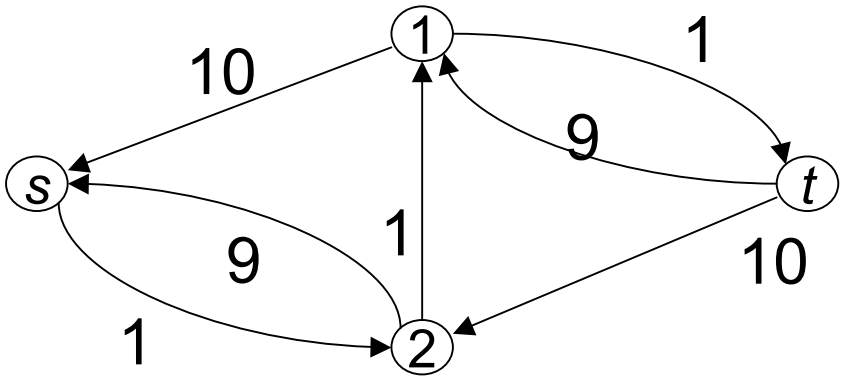
Beispiel





Beispiel





Maximale Flüsse

Lemma 2: Sei *f* ein zulässiger (*s,t*)-Fluss in *N* und sei $SN = (\overline{V}, \overline{E}, \overline{c})$ das Schichtnetzwerk bzgl. *f*.

a) *f* ist ein maximaler Fluss gdw. $t \notin \overline{V}$

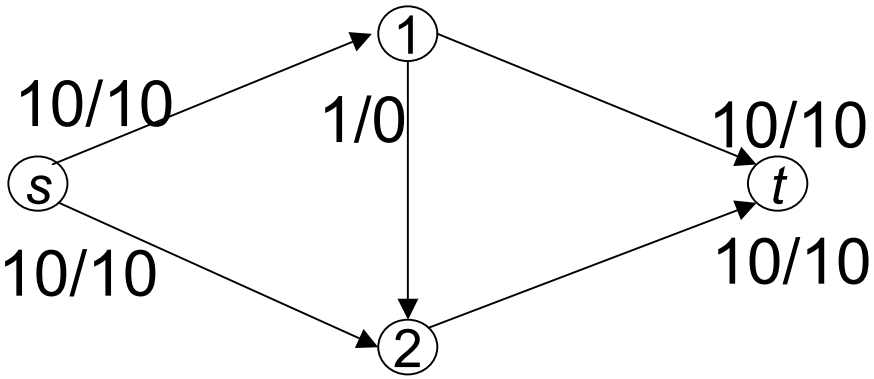
b) Sei $\overline{f}$ ein zulässiger (*s,t*)-Fluss in S*N*. Dann ist *f ´*: *E* → *R* mit

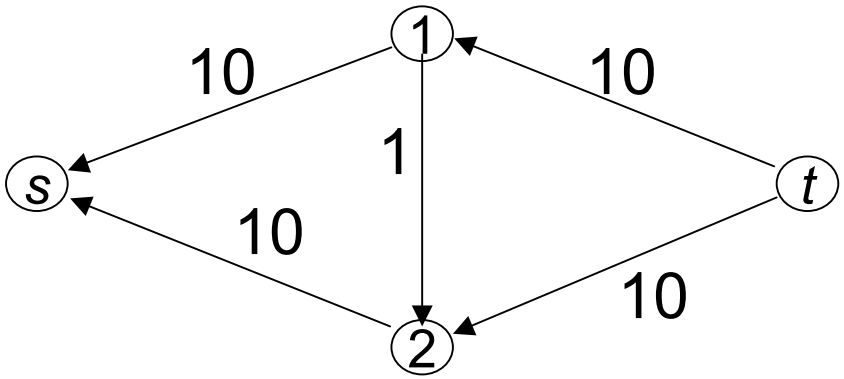
$$f'(e) = f(e) + \bar{f}(e_1) - \bar{f}(e_2)$$

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

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

Beispiel





Beweis, Teil b)

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

Beweis, Teil b)

Für jeden Knoten $v \in V$ gilt:

$$\sum_{e \in aus(v)} f'(e) - \sum_{e \in ein(v)} f'(e)$$

$$=$$

Flusserhaltung :

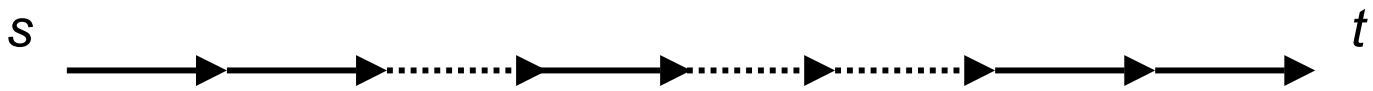
Wert :

Beweis, Teil a)

a) "$\Rightarrow$"

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

Dann existiert ein Weg *P* von *s* nach *t* in S*N*.



ε = min. Kapazität von Kanten in *P*

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

Also ist *f* nicht maximal.

Beweis, Teil a)

"$\Leftarrow$"

Setze $S = \overline{V}, \quad T = V - S$

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

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

$f(e) = c(e)$ für $e \in S \times T$

$f(e) = 0$ 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.

Maximale Flüsse und minimale Schnitte

Satz 1: Seien $N = (V, E, c)$ ein Netzwerk und $s, t \in V$.

f_{max}= max. Wert eines zulässigen (s,t)-Flusses

c_{min} = min. Kapazität eines (s,t)-Schnittes.

$$f_{max} = c_{min}$$

Beweis: f_{max} und c_{min} existieren. Wegen Lemma 1 gilt $f_{max} \leq c_{min}$

Seien f ein Fluss mit $W(f) = f_{max}$ und $SN = (\overline{V},\overline{E},\overline{c})$ das Schichtnetzwerk bzgl. f.

Setze $S = \overline{V}$ und $T = V - S$. Im Beweis von Lemma 2 zeigten wir:

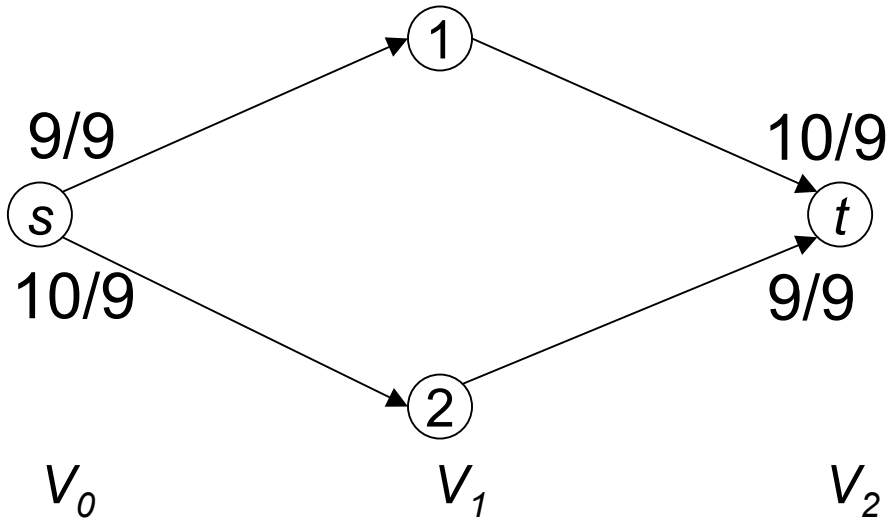
$$W(f) = \sum_{e \in E \cap (S \times T)} f(e) - \sum_{e \in E \cap (T \times S)} f(e) = C(S,T)$$

5. Blockierende Flüsse

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

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

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



Algorithmus

1. $f(e) \leftarrow 0$ für alle $e \in E$;
2. Konstruiere Schichtnetzwerk $SN = (\overline{V},\overline{E},\overline{c})$ bzgl. f;
3. **while** $t \in \overline{V}$ **do**
4. Finde einen blockierenden Fluss $\overline{f}$ in SN;
5. Aktualisiere f gemäß $\overline{f}$ 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 3: Sei k_i die Tiefe des Schichtnetzwerks in der *i*-ten Iteration.
Dann gilt $k_i > k_{i-1}$, für $i \geq 2$.

Beweis: Schichtnetzwerk in der *i*-ten Iteration: SN_i
Es existiert Weg *P* von *s* to *t* der Länge k_i.

$$s = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \xrightarrow{e_3} \ldots\ldots \xrightarrow{e_{k_{i-1}}} v_{k_{i-1}} \xrightarrow{e_{k_i}} v_{k_i} = t$$

d_j = Schichtnummer von v_j in SN_{i-1}, $0 \leq j \leq k_i$
$d_j = \infty$ wenn v_j kein Knoten in SN_{i-1}

Die Tiefe eines Schichtnetzwerks

Behauptung:

Für alle $i \geq 2$ gilt:

a) Gibt es eine Kante von v_{j-1} nach v_j in SN_{i-1}, dann gilt $d_j = d_{j-1} + 1$.

b) Gibt es keine Kante von v_{j-1} nach v_j in SN_{i-1} dann gilt $d_j \leq d_{j-1}$.

c) $k_{i-1} < k_i$

Beweis:

a) Offensichtlich.

Teil b)

b) Annahme: $d_j \geq d_{j-1} + 1$

f_{i-1} ergibt SN_{i-1} f_i ergibt SN_i

$$\text{Ist } (v_{j-1}, v_j) \in E \Rightarrow$$

$$\text{Ist } (v_j, v_{j-1}) \in E \Rightarrow$$

Also $(v_j, v_{j-1}) \in \overline{E}_{i-1}$

Somit ist $d_{j-1} = d_j + 1$ und $d_j = d_{j-1} - 1 < d_{j-1}$

Teil c)

c) Da $v_0 = s$ und $d_0 = 0$, folgt aus a) und b), dass $d_j \leq j$, für $1 \leq j \leq k_i$.

Somit $k_{i-1} = d_{k_i} \leq k_i$.

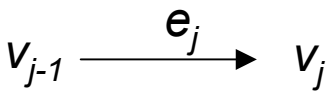
Annahme: $k_{i-1} = k_i$.

Dann existiert ein Weg P in LN_{i-1}.

$$s = v_0 \xrightarrow{e_1} v_1 \xrightarrow{e_2} v_2 \xrightarrow{e_3} \ldots\ldots \xrightarrow{e_{k_{i-1}}} v_{k_{i-1}} \xrightarrow{e_{k_i}} v_{k_i} = t$$

Teil c)

Da wir einen blockierenden Fluss in SN_{i-1} berechnen, gibt es eine gesättigte Kante

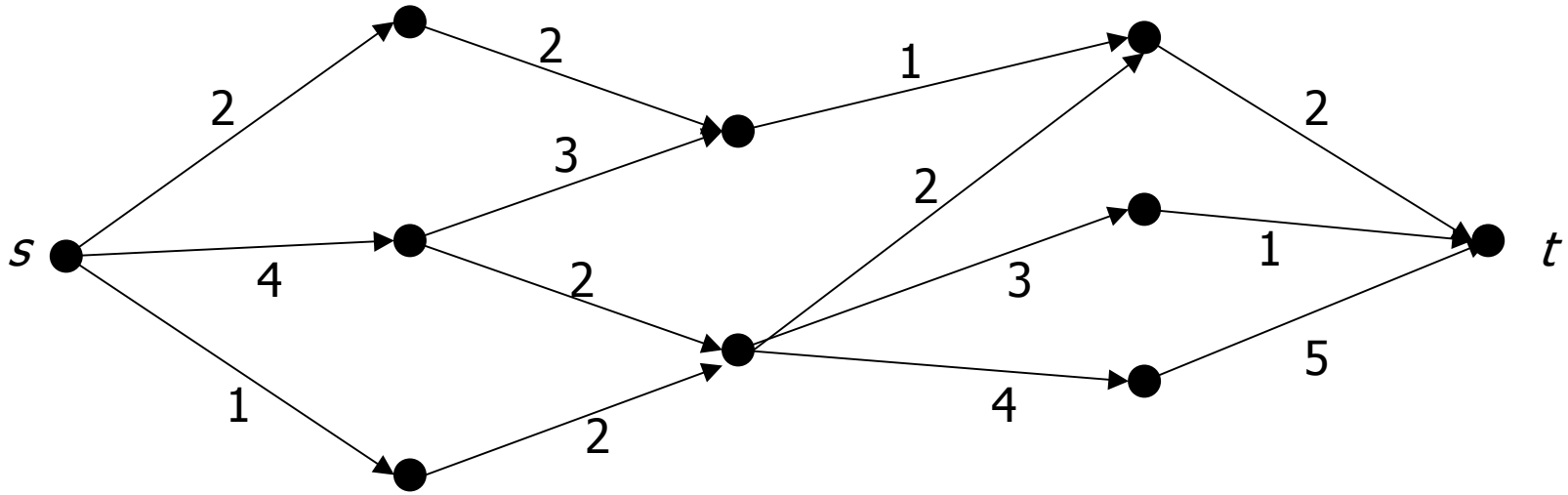


$$(v_{j-1}, v_j) \in E \Rightarrow$$
$$(v_j, v_{j-1}) \in E \Rightarrow$$

Die Anzahl der Iterationen

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.

Analyse

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

Beweis: k =Tiefe des Schichtnetzwerks

Konstruktion eines Wegs benötigt Zeit
$O(k$ + # 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*

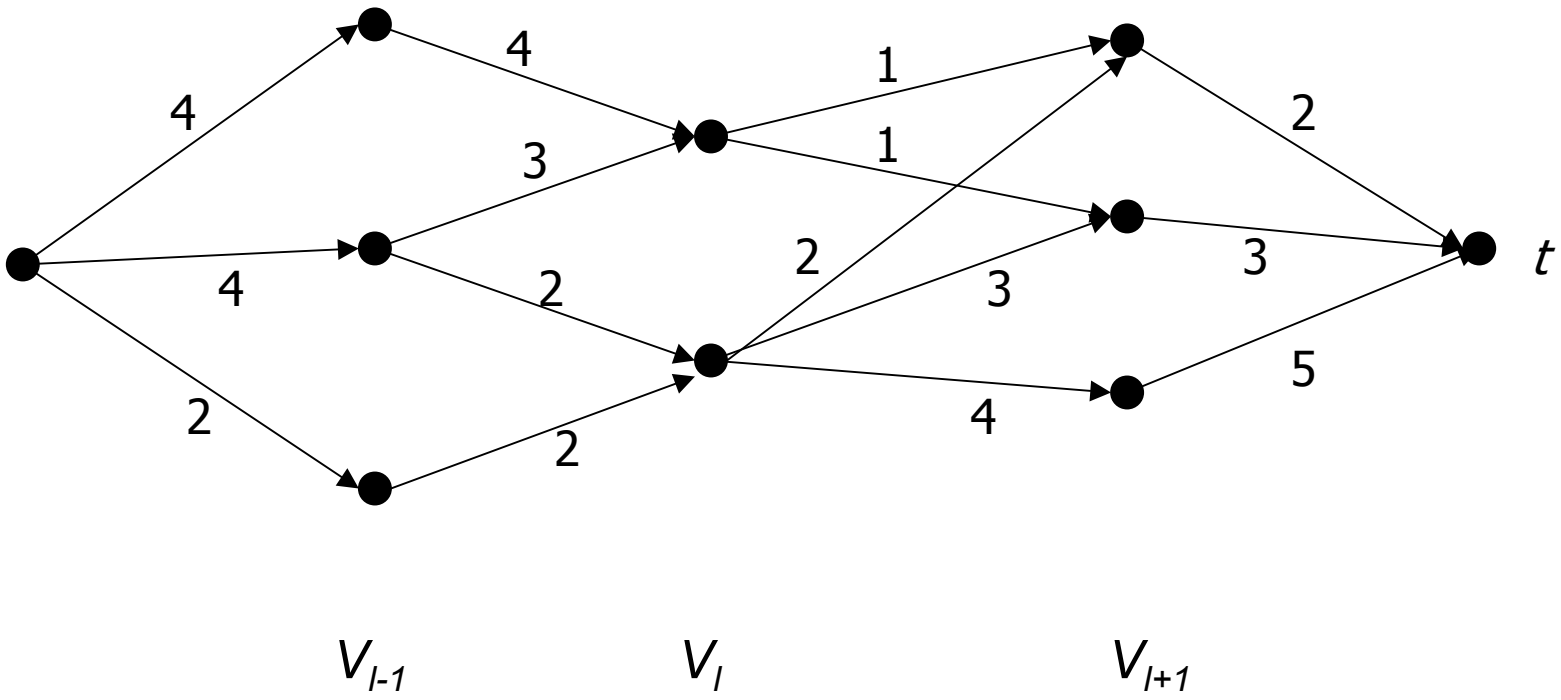
$$PO(v) = \min\left\{ \sum_{e \in aus(v)} c(e) - f(e), \sum_{e \in ein(v)} c(e) - f(e) \right\}$$

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

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);

\\ 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, vermindere $c(e)$ 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** $c(e) = 0$ **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, $0 \le l \le k$, **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.

8. *d*-beschränkte Netzwerke

Definition: Sei d eine natürliche Zahl. $N = (V,E,c)$ ist *d*-beschränkt, wenn $c(e) \in \{1,2,...,d\}$ für alle $e \in E$.

1-beschränkte Netzwerke heißen (0,1)-Netzwerke.

Anwendung unserer Flussalgorithmen auf *d*-beschränkte Netzwerke:

→ alle berechneten Flüsse sind ganzzahlig, d.h. $f(e) \in IN_0$

→ der maximale Fluss ist ganzzahlig

d-beschränkte Netzwerke

Satz 5: Ein blockierender Fluss kann in einem *d*-beschränkten Netzwerk in Zeit $O(de)$ berechnet werden. Für $d = 1$ ergibt sich Zeit $O(e)$.

Beweis: DFS-Algorithmus

Zeit für die Konstruktion eines Wegs:

O(# Kanten auf s-t-Weg + # durchlaufenen Kanten, die in Sackgassen enden)

Jede Kante ist in höchstens d Wegen enthalten.

Maximale Flüsse in Restnetzwerken

Lemma 4: Seien N ein Netzwerk und f_{max} der Wert eines maximalen (s,t)-Flusses. Seien RN das Restnetzwerk bzgl. eines Flusses f und $\overline{f}_{max}$ der Wert eines maximalen (s,t)-Flusses in RN. Dann gilt

$$f_{max} = \overline{f}_{max} + W(f).$$

Beweis: Sei $(S, V - S)$ ein (s,t)-Schnitt.

$\underline{C}(S, V - S)$: Kapazität von $(S, V - S)$ bzgl. N

$\overline{C}(S, V - S)$: Kapazität von $(S, V - S)$ bzgl. RN

Also

$$\overline{C}_{\min} = C_{\min} - W(f),$$

wobei $C_{\min}, \overline{C}_{\min}$ die minimalen Kapazitäten von $(s,t)-$Schnitten in N bzw. RN sind.

9. Einfache Netzwerke

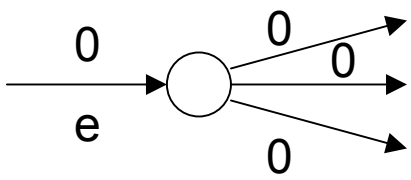
Definition: Ein Netzwerk $N = (V,E,c)$ ist einfach, wenn $indeg(v) = 1$ oder $outdeg(v) = 1$ für alle $v \in V$.

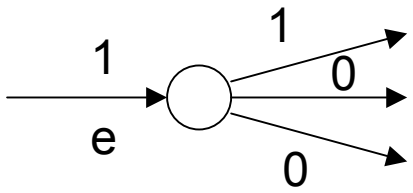
Satz 6: Sei $N = (V,E,c)$ ein einfaches (0,1)-Netzwerk. Dann kann ein maximaler Fluss in Zeit $O(n^{1/2}e)$ berechnet werden.

Restnetzwerke einfacher Netzwerke

Behauptung: Sei N ein einfaches Netzwerk und f ein Fluss in N.
Dann ist RN einfach.

Beweis: Sei $v \in V$ und $indeg(v) = 1$ ($outdeg(v) = 1$ analog).
Ist $f(e) = 0$ für $e \in ein(v)$, dann ist $f(e') = 0$ für alle $e' \in aus(v)$,
und v hat Eingangsgrad 1 in RN.





Ist $f(e) = 1$ für $e \in ein(v)$, dann ist $f(e') = 1$ für genau ein $e' \in aus(v)$,
und v hat Eingangsgrad 1 in RN.

Beweis von Satz 6

Blockierender Fluss kann in Zeit $O(e)$ berechnet werden.

Wir zeigen: # Iterationen = $O(n^{1/2})$.

f_{max} = Wert eines maximalen (s,t)-Flusses.

$f_{max} < n^{1/2}$: ok

Angenommen $f_{max} \geq n^{1/2}$.

Sei Iteration l diejenige, in der der Flusswert auf $\geq f_{max} - n^{1/2}$ ansteigt.

Wir zeigen, dass das Schichtnetzwerk in Iteration l Tiefe $\leq n^{1/2}$ hat.

f : zulässiger (s,t)-Fluss unmittelbar vor Iteration l

RN: Restnetzwerk bzgl. f.

Beweis von Satz 6

Wegen Lemma 4 gibt es in *RN* einen Fluss $\bar{f}$ mit Wert

$$\bar{f}_{\max} = f_{\max} - W(f) \geq f_{\max} - \left(f_{\max} - n^{1/2}\right) = n^{1/2}$$

O.B.d.A. ist $\bar{f}$ ganzzahlig, d.h. $\bar{f}(e) \in \{0,1\}$.
RN ist einfach, und daher läuft höchstens eine Flusseinheit durch jeden Knoten $v \in V - \{s,t\}$.

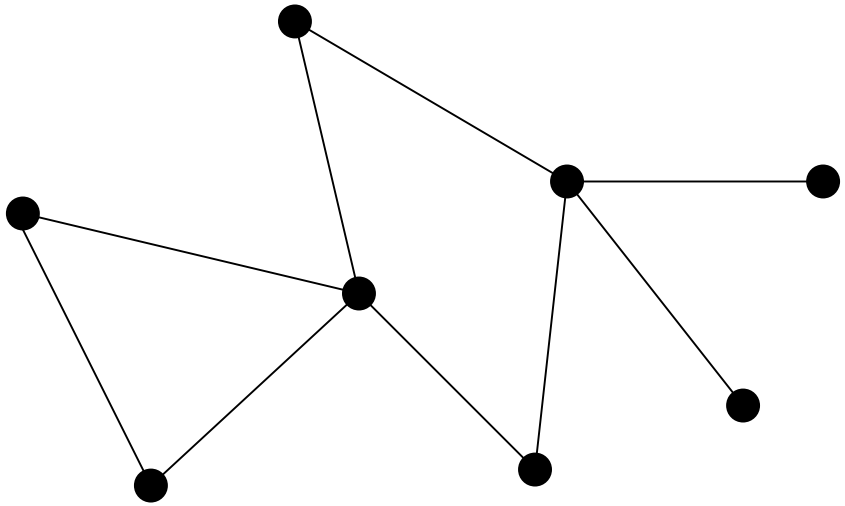
$\bar{f}$ besteht aus $n^{1/2}$ knotendisjunkten Wegen von *s* nach *t*.
Also gibt es einen Weg mit $< n^{1/2}$ Zwischenknoten.

10. Matchings in bipartiten Graphen

$G = (V,E)$ ungerichteter Graph.

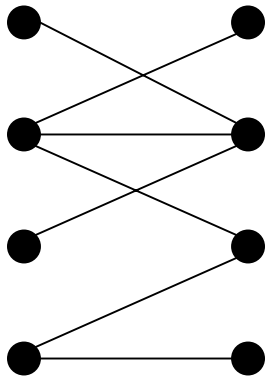
Matching M ist eine Kantenmenge $M \subseteq E$, so dass keine zwei Kanten $e_1, e_2 \in M$, $e_1 \neq e_2$, einen gemeinsamen Endknoten haben.

Ein maximales Matching ist ein Matching maximaler Kardinalität.



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 7: 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 $O(n^{1/2}e)$ berechnet werden.

Beweis: Konstruiere einfaches Netzwerk wie folgt:

(Alle Kapazitäten sind 1.)