Primzahltest

n	**2**	**3**	4	**5**	6	**7**	8	9	10	**11**	12	**13**	14	15	16	**17**	18	**19**	20	21	22	**23**	...
$f(n)$	0	1	0	1	2	1	0	4	2	1	8	1	2	4	0	1	14	1	8	4	2	1	...

99131 prim?

123456789011 prim?

12345236789876453425434509935647111898751324098654378906435423 1 prim?

Primzahlfunktion

$$2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, \ldots$$

$$\text{Primzahlfunktion} \quad \pi(x) := |\{n \in \mathbb{N} \mid n \leq x, n \text{ prim}\}| \sim \frac{x}{\ln x}$$

Primzahlfunktion

2, **3**, 4, **5**, 6, **7**, 8, 9, 10, **11**, 12, **13**, 14, 15, 16, **17**, 18, **19**, 20, 21, 22, **23**, ...

$$\text{Primzahlfunktion} \quad \pi(x) := |\{n \in \mathbb{N} \mid n \leq x, n \text{ prim}\}| \sim \frac{x}{\ln x}$$

Prüfe mit Target-Shooting!

Primzahlfunktion

2, **3**, 4, **5**, 6, **7**, 8, 9, 10, **11**, 12, **13**, 14, 15, 16, **17**, 18, **19**, 20, 21, 22, **23**, ...

Primzahlfunktion $\pi(x) := |\{n \in \mathbb{N} \mid n \leq x, n \text{ prim}\}| \sim \frac{x}{\ln x}$

Prüfe mit Target-Shooting!
Dazu brauchen wir einen Primzahltest!

Problemstellung

Gegeben seien eine Zahl $n \in \mathbb{N}$, entscheide, ob n prim ist, d.h. keinen Teiler in $\{2, \ldots, n-1\}$ hat.

Problemstellung

Gegeben seien eine Zahl $n \in \mathbb{N}$, entscheide, ob n prim ist, d.h. keinen Teiler in $\{2, \ldots, n-1\}$ hat.

Anwendung: Erzeugen von zufälligen grossen Primzahlen für RSA-Kryptographie (z.B. 200 Bits).

Problemstellung

Gegeben seien eine Zahl $n \in \mathbb{N}$, entscheide, ob n prim ist, d.h. keinen Teiler in $\{2, \ldots, n-1\}$ hat.

Anwendung: Erzeugen von zufälligen grossen Primzahlen für RSA-Kryptographie (z.B. 200 Bits).

Dazu erzeugen wir zufällig Zahlen n mit 200 Bits (d.h. $n \approx 2^{200}$) und testen sie auf prim. Der Primzahlensatz garantiert uns eine hohe Erfolgsrate (etwa $\frac{1}{200 \ln 2} \approx \frac{1}{139}$).

Problemstellung

Gegeben seien eine Zahl $n \in \mathbb{N}$, entscheide, ob n prim ist, d.h. keinen Teiler in $\{2, \ldots, n-1\}$ hat.

Anwendung: Erzeugen von zufälligen grossen Primzahlen für RSA-Kryptographie (z.B. 200 Bits).

Dazu erzeugen wir zufällig Zahlen n mit 200 Bits (d.h. $n \approx 2^{200}$) und testen sie auf prim. Der Primzahlensatz garantiert uns eine hohe Erfolgsrate (etwa $\frac{1}{200 \ln 2} \approx \frac{1}{139}$).

Probieren aller Zahlen bis $\sqrt{n} \approx 2^{100}$ ist zu langsam. Wir wollen einen Algorithmus polynomiell in $\log n$ (Darstellungsgrösse von n).

Grösster gemeinsamer Teiler

Für $m, n \in \mathbb{Z}$, sei $\mathrm{ggT}(m, n)$ der grösste gemeinsame Teiler von m und n. Er kann mit Hilfe des Euklid'schen Algorithmus schnell berechnet werden (in $O((\log nm)^3)$). Natürlich gilt:

$$\mathrm{ggT}(a, n) > 1 \text{ für } a \in [n-1] \quad \Rightarrow \quad n \text{ nicht prim}$$

Euklid-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** $\mathrm{ggT}(a, n) > 1$ **then return** 'keine Primzahl'
3: **else return** 'Primzahl'

Grösster gemeinsamer Teiler

Für $m, n \in \mathbb{Z}$, sei $\mathrm{ggT}(m, n)$ der grösste gemeinsame Teiler von m und n. Er kann mit Hilfe des Euklid'schen Algorithmus schnell berechnet werden (in $O((\log nm)^3)$). Natürlich gilt:

$$\mathrm{ggT}(a, n) > 1 \text{ für } a \in [n-1] \quad \Rightarrow \quad n \text{ nicht prim}$$

Euklid-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** $\mathrm{ggT}(a, n) > 1$ **then return** 'keine Primzahl'
3: **else return** 'Primzahl'

- Ausgabe 'keine Primzahl' ist immer richtig.

Grösster gemeinsamer Teiler

Für $m, n \in \mathbb{Z}$, sei $\mathrm{ggT}(m, n)$ der grösste gemeinsame Teiler von m und n. Er kann mit Hilfe des Euklid'schen Algorithmus schnell berechnet werden (in $O((\log nm)^3)$). Natürlich gilt:

$$\mathrm{ggT}(a, n) > 1 \text{ für } a \in [n-1] \quad \Rightarrow \quad n \text{ nicht prim}$$

Euklid-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** $\mathrm{ggT}(a, n) > 1$ **then return** 'keine Primzahl'
3: **else return** 'Primzahl'

- Ausgabe 'keine Primzahl' ist immer richtig.
- Falls n nicht prim: Falsche Ausgabe 'Primzahl' mit W'keit $\frac{|\mathbb{Z}_n^*|}{n-1}$.

Grösster gemeinsamer Teiler

Für $m, n \in \mathbb{Z}$, sei $\mathrm{ggT}(m, n)$ der grösste gemeinsame Teiler von m und n. Er kann mit Hilfe des Euklid'schen Algorithmus schnell berechnet werden (in $O((\log nm)^3)$). Natürlich gilt:

$$\mathrm{ggT}(a, n) > 1 \text{ für } a \in [n-1] \quad \Rightarrow \quad n \text{ nicht prim}$$

Euklid-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** $\mathrm{ggT}(a, n) > 1$ **then return** 'keine Primzahl'
3: **else return** 'Primzahl'

- Ausgabe 'keine Primzahl' ist immer richtig.
- Falls n nicht prim: Falsche Ausgabe 'Primzahl' mit W'keit $\frac{|\mathbb{Z}_n^*|}{n-1}$.

$\mathrm{ggT}(\mathbf{1}, 9) = \mathrm{ggT}(\mathbf{2}, 9) = \mathrm{ggT}(\mathbf{4}, 9) = \mathrm{ggT}(\mathbf{5}, 9) = \mathrm{ggT}(\mathbf{7}, 9) = \mathrm{ggT}(\mathbf{8}, 9) = 1$

$\mathbb{Z}_n^*$ (multiplikative Gruppe mod n)

$$\mathrm{ggT}(\mathbf{1}, 9) = \mathrm{ggT}(\mathbf{2}, 9) = \mathrm{ggT}(\mathbf{4}, 9) = \mathrm{ggT}(\mathbf{5}, 9) = \mathrm{ggT}(\mathbf{7}, 9) = \mathrm{ggT}(\mathbf{8}, 9) = 1$$

$$\mathbb{Z}_n^* = \{a \in [n-1] \mid \mathrm{ggT}(a, n) = 1\}; \qquad \underbrace{\varphi(n)}_{\text{eulersche Phi-Funktion}} := |\mathbb{Z}_n^*|$$

$\mathbb{Z}_n^*$ (multiplikative Gruppe mod n)

$$\mathrm{ggT}(\mathbf{1},9) = \mathrm{ggT}(\mathbf{2},9) = \mathrm{ggT}(\mathbf{4},9) = \mathrm{ggT}(\mathbf{5},9) = \mathrm{ggT}(\mathbf{7},9) = \mathrm{ggT}(\mathbf{8},9) = 1$$

$$\mathbb{Z}_n^* = \{a \in [n-1] \mid \mathrm{ggT}(a,n) = 1\}; \qquad \underbrace{\varphi(n)}_{\text{eulersche Phi-Funktion}} := |\mathbb{Z}_n^*|$$

$$\mathbb{Z}_9^* = \{1,2,4,5,7,8\},\ \varphi(9) = 6. \quad \mathbb{Z}_7^* = \{1,2,3,4,5,6\},\ \varphi(7) = 6.$$

$\mathbb{Z}_n^*$ (multiplikative Gruppe mod n)

$$\mathrm{ggT}(\mathbf{1},9) = \mathrm{ggT}(\mathbf{2},9) = \mathrm{ggT}(\mathbf{4},9) = \mathrm{ggT}(\mathbf{5},9) = \mathrm{ggT}(\mathbf{7},9) = \mathrm{ggT}(\mathbf{8},9) = 1$$

$$\mathbb{Z}_n^* = \{a \in [n-1] \mid \mathrm{ggT}(a,n) = 1\}; \qquad \underbrace{\varphi(n)}_{\text{eulersche Phi-Funktion}} := |\mathbb{Z}_n^*|$$

$\mathbb{Z}_9^* = \{1,2,4,5,7,8\}$, $\varphi(9) = 6$. $\quad \mathbb{Z}_7^* = \{1,2,3,4,5,6\}$, $\varphi(7) = 6$.

n prim: $\mathbb{Z}_n^* = [n-1]$ und $\varphi(n) = n-1$.

$\mathbb{Z}_n^*$ (multiplikative Gruppe mod n)

$$\mathrm{ggT}(\mathbf{1},9) = \mathrm{ggT}(\mathbf{2},9) = \mathrm{ggT}(\mathbf{4},9) = \mathrm{ggT}(\mathbf{5},9) = \mathrm{ggT}(\mathbf{7},9) = \mathrm{ggT}(\mathbf{8},9) = 1$$

$$\mathbb{Z}_n^* = \{a \in [n-1] \mid \mathrm{ggT}(a,n) = 1\}; \qquad \underbrace{\varphi(n)}_{\text{eulersche Phi-Funktion}} := |\mathbb{Z}_n^*|$$

$\mathbb{Z}_9^* = \{1,2,4,5,7,8\}$, $\varphi(9) = 6$. $\quad \mathbb{Z}_7^* = \{1,2,3,4,5,6\}$, $\varphi(7) = 6$.

n prim: $\mathbb{Z}_n^* = [n-1]$ und $\varphi(n) = n-1$.

$n = p^2$, p prim: $\varphi(n) = p(p-1) = n - \sqrt{n}$ $(\Rightarrow \frac{|\mathbb{Z}_n^*|}{n-1} \approx 1 - \frac{1}{\sqrt{n}})$.

$\mathbb{Z}_n^*$ (multiplikative Gruppe mod n)

$$\mathrm{ggT}(1,9) = \mathrm{ggT}(2,9) = \mathrm{ggT}(4,9) = \mathrm{ggT}(5,9) = \mathrm{ggT}(7,9) = \mathrm{ggT}(8,9) = 1$$

$$\mathbb{Z}_n^* = \{a \in [n-1] \mid \mathrm{ggT}(a,n) = 1\}; \qquad \underbrace{\varphi(n)}_{\text{eulersche Phi-Funktion}} := |\mathbb{Z}_n^*|$$

$\mathbb{Z}_9^* = \{1,2,4,5,7,8\}$, $\varphi(9) = 6$. $\quad \mathbb{Z}_7^* = \{1,2,3,4,5,6\}$, $\varphi(7) = 6$.

n prim: $\mathbb{Z}_n^* = [n-1]$ und $\varphi(n) = n-1$.

$n = p^2$, p prim: $\varphi(n) = p(p-1) = n - \sqrt{n}$ $(\Rightarrow \frac{|\mathbb{Z}_n^*|}{n-1} \approx 1 - \frac{1}{\sqrt{n}})$.

$\mathbb{Z}_n^*$ mit Multiplikation mod n bildet eine Gruppe
(der Ordnung $\varphi(n)$).

In $\mathbb{Z}_9^*$: $2 \cdot 7 = 5$, $\quad 2 \cdot 5 = 1$, $\quad 2^{-1} = 5$, $\quad 4^2 = 7$, $\quad 4^3 = 1$, $\quad \ldots$

Erinnerung Gruppentheorie

- Satz von Lagrange $\Rightarrow$ Ist $H \leq G$ (H Untergruppe von G), dann ist $|H|$ ein Teiler von $|G|$.
- $\mathrm{ord}(a) := \min\{i \in \mathbb{N} \mid a^i = 1\}$, die Ordnung der von a erzeugten Untergruppe $\langle a \rangle := \{a^0 = 1, a^1, a^2, \ldots\}$.
- $\mathrm{ord}(a)$ ist Teiler von $|G|$. Folglich gilt

$$a^{|G|} = a^{\mathrm{ord}(a)\frac{|G|}{\mathrm{ord}(a)}} = 1^{\frac{|G|}{\mathrm{ord}(a)}} = 1.$$

- In $\mathbb{Z}_n^*$: $a^{\varphi(n)} = 1$ für alle $a \in \mathbb{Z}_n^*$ (weil $\varphi(n) = |\mathbb{Z}_n^*|$)
- In $\mathbb{Z}_n^*$, n prim: $a^{n-1} = 1$ für alle $a \in \mathbb{Z}_n^*$ (weil $\varphi(n) = n - 1$).

Satz (Kleiner fermatscher Satz)

Ist $n \in \mathbb{N}$ prim, so gilt für alle Zahlen $a \in [n-1]$

$$a^{n-1} \equiv 1 \pmod{n} \qquad \textit{(bzw. } a^{n-1} = 1 \textit{ in } \mathbb{Z}_n^*\textit{)}.$$

Primzahltest nach kleinem Satz von Fermat

Fermat-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** ggT$(a, n) > 1$ oder $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
3: **return** 'keine Primzahl'
4: **else**
5: **return** 'Primzahl'

"$a^{n-1} \not\equiv 1 \pmod{n}$?" schnell mit binärer Exponentiation.

Primzahltest nach kleinem Satz von Fermat

Fermat-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** ggT$(a, n) > 1$ oder $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
3: **return** 'keine Primzahl'
4: **else**
5: **return** 'Primzahl'

"$a^{n-1} \not\equiv 1 \pmod{n}$?" schnell mit binärer Exponentiation.

- Die Ausgabe 'keine Primzahl' ist immer richtig.

Primzahltest nach kleinem Satz von Fermat

Fermat-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** ggT$(a, n) > 1$ oder $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
3: **return** 'keine Primzahl'
4: **else**
5: **return** 'Primzahl'

"$a^{n-1} \not\equiv 1 \pmod{n}$?" schnell mit binärer Exponentiation.

- Die Ausgabe 'keine Primzahl' ist immer richtig.
- Die Ausgabe 'Primzahl' ist falsch mit W'keit $< \frac{1}{2}$, es sei denn n ist eine Carmichael-Zahl (Def. später).

Primzahltest nach kleinem Satz von Fermat

Fermat-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** ggT$(a, n) > 1$ oder $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
3: **return** 'keine Primzahl'
4: **else**
5: **return** 'Primzahl'

"$a^{n-1} \not\equiv 1 \pmod{n}$?" schnell mit binärer Exponentiation.

- Die Ausgabe 'keine Primzahl' ist immer richtig.
- Die Ausgabe 'Primzahl' ist falsch mit W'keit $< \frac{1}{2}$, es sei denn n ist eine Carmichael-Zahl (Def. später).

Durch k-maliges Wiederholen im Fall 'Primzahl' können wir die Fehlerwahrscheinlichkeit auf $\leq \frac{1}{2^k}$ drücken.

Primzahltest nach kleinem Satz von Fermat

Fermat-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** ggT$(a, n) > 1$ oder $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
3: **return** 'keine Primzahl'
4: **else**
5: **return** 'Primzahl'

"$a^{n-1} \not\equiv 1 \pmod{n}$?" schnell mit binärer Exponentiation.

- Die Ausgabe 'keine Primzahl' ist immer richtig.
- Die Ausgabe 'Primzahl' ist falsch mit W'keit $< \frac{1}{2}$, es sei denn n ist eine Carmichael-Zahl (Def. später).

Wir machen einen Fehler, falls

$$n \text{ nicht prim, ggT}(a, n) = 1 \text{ und } a^{n-1} \equiv 1 \pmod{n}$$

Ein solches a nennen wir Pseudoprimzahlbasis von n.

Die ominöse Funktion f

n	**2**	**3**	4	**5**	6	**7**	8	9	10	**11**	12	**13**	14	15	16	**17**	18	**19**	20	21	22	**23**	...
$\underbrace{f(n)}_{2^{n-1} \bmod n}$	0	1	0	1	2	1	0	4	2	1	8	1	2	4	0	1	14	1	8	4	2	1	...

Die ominöse Funktion f

n	**2**	**3**	4	**5**	6	**7**	8	9	10	**11**	12	**13**	14	15	16	**17**	18	**19**	20	21	22	**23**	...
$\underbrace{f(n)}_{2^{n-1} \bmod n}$	0	1	0	1	2	1	0	4	2	1	8	1	2	4	0	1	14	1	8	4	2	1	...

Die ersten nicht primen Zahlen mit $2^{n-1} \bmod n = 1$:

341, 561, 645, 1105, 1387, 1729, ...

Pseudoprimzahlbasen und Carmichael-Zahlen

$$
\begin{aligned}
PB_n &:= \{a \in [n-1] \mid \mathrm{ggT}(a,n) = 1 \text{ und } a^{n-1} \equiv 1 \pmod{n}\} \\
&= \{a \in \mathbb{Z}_n^* \mid a^{n-1} \equiv 1 \pmod{n}\} \quad \text{Pseudoprimzahlbasen}
\end{aligned}
$$

Pseudoprimzahlbasen und Carmichael-Zahlen

$$\begin{aligned} PB_n &:= \{a \in [n-1] \mid \text{ggT}(a,n) = 1 \text{ und } a^{n-1} \equiv 1 \pmod{n}\} \\ &= \{a \in \mathbb{Z}_n^* \mid a^{n-1} \equiv 1 \pmod{n}\} \quad \text{Pseudoprimzahlbasen} \end{aligned}$$

n heisst Carmichael-Zahl, falls n nicht prim ist und $PB_n = \mathbb{Z}_n^*$.
kleinste Beispiele: $561 = 3 \cdot 11 \cdot 17, 1105 = 5 \cdot 13 \cdot 17, 1729 = 7 \cdot 13 \cdot 19, \ldots$

Pseudoprimzahlbasen und Carmichael-Zahlen

$$\begin{aligned} PB_n &:= \{a \in [n-1] \mid \mathrm{ggT}(a,n) = 1 \text{ und } a^{n-1} \equiv 1 \pmod{n}\} \\ &= \{a \in \mathbb{Z}_n^* \mid a^{n-1} \equiv 1 \pmod{n}\} \quad \text{Pseudoprimzahlbasen} \end{aligned}$$

n heisst Carmichael-Zahl, falls n nicht prim ist und $PB_n = \mathbb{Z}_n^*$.
kleinste Beispiele: $561 = 3 \cdot 11 \cdot 17, 1105 = 5 \cdot 13 \cdot 17, 1729 = 7 \cdot 13 \cdot 19, \ldots$

PB_n ist Untergruppe von $\mathbb{Z}_n^*$, weil

$$\begin{aligned} a^{n-1} \equiv 1 \pmod{n} \quad &\wedge \quad b^{n-1} \equiv 1 \pmod{n} \\ &\Rightarrow \quad (ab \bmod n)^{n-1} \equiv 1 \pmod{n} \ . \end{aligned}$$

Pseudoprimzahlbasen und Carmichael-Zahlen

$$\begin{aligned} PB_n &:= \{a \in [n-1] \mid \mathrm{ggT}(a,n) = 1 \text{ und } a^{n-1} \equiv 1 \pmod{n}\} \\ &= \{a \in \mathbb{Z}_n^* \mid a^{n-1} \equiv 1 \pmod{n}\} \quad \text{Pseudoprimzahlbasen} \end{aligned}$$

n heisst Carmichael-Zahl, falls n nicht prim ist und $PB_n = \mathbb{Z}_n^*$.
kleinste Beispiele: $561 = 3 \cdot 11 \cdot 17, 1105 = 5 \cdot 13 \cdot 17, 1729 = 7 \cdot 13 \cdot 19, \ldots$

PB_n ist Untergruppe von $\mathbb{Z}_n^*$, weil

$$\begin{aligned} a^{n-1} \equiv 1 \pmod{n} \quad &\wedge \quad b^{n-1} \equiv 1 \pmod{n} \\ &\Rightarrow \quad (ab \bmod n)^{n-1} \equiv 1 \pmod{n} \ . \end{aligned}$$

Folglich, wenn $PB_n \neq \mathbb{Z}_n^*$ (d.h. n ist nicht Carmichael), dann ist $|PB_n|$ ein echter Teiler von $|\mathbb{Z}_n^*|$, und $|PB_n| \leq \frac{\varphi(n)}{2} < \frac{n-1}{2}$.

Primzahltest nach kleinem Satz von Fermat

Fermat-Primzahltest(n)

1: Wähle $a \in [n-1]$, zufällig gleichverteilt
2: **if** ggT$(a, n) > 1$ oder $a^{n-1} \not\equiv 1 \pmod{n}$ **then**
3: **return** 'keine Primzahl'
4: **else**
5: **return** 'Primzahl'

- Die Ausgabe 'keine Primzahl' ist immer richtig.
- Die Ausgabe 'Primzahl' ist falsch wenn

  n nicht prim, ggT$(a, n) = 1$ und $a^{n-1} \equiv 1 \pmod{n}$

  d.h. mit W'keit $\frac{|PB_n|}{n-1} < \frac{1}{2}$,

  es sei denn, n ist eine Carmichael-Zahl.

Zertifikate für 'nicht prim'

Für "n nicht prim", ist $a \in [n-1]$

- triviales Zertifikat, falls $a \geq 2$ und $a|n$ (a Teiler von n).

- Euklid Zertifikat, falls $\text{ggT}(a, n) > 1$.
- Fermat Zertifikat, falls $a^{n-1} \bmod n \neq 1$.

Zertifikate für 'nicht prim'

Für "n nicht prim", ist $a \in [n-1]$

- triviales Zertifikat, falls $a \geq 2$ und $a|n$ (a Teiler von n).
  Anteil $> \frac{1}{n}$ (aber $a \in [\lfloor\sqrt{n}\rfloor]$ genügt).
- Euklid Zertifikat, falls $\mathrm{ggT}(a, n) > 1$. Anteil $> \frac{1}{\sqrt{n}}$
- Fermat Zertifikat, falls $a^{n-1} \bmod n \neq 1$.
  Anteil $> \frac{1}{2}$, ausser n ist Carmichael.

Zertifikate für 'nicht prim'

Für "n nicht prim", ist $a \in [n-1]$

- triviales Zertifikat, falls $a \geq 2$ und $a|n$ (a Teiler von n).
  Anteil $> \frac{1}{n}$ (aber $a \in [\lfloor\sqrt{n}\rfloor]$ genügt).
- Euklid Zertifikat, falls $\mathrm{ggT}(a, n) > 1$. Anteil $> \frac{1}{\sqrt{n}}$
- Fermat Zertifikat, falls $a^{n-1} \bmod n \neq 1$.
  Anteil $> \frac{1}{2}$, ausser n ist Carmichael.
- Miller-Rabin Zertifikat, falls ... (folgt).
  Anteil $> \frac{3}{4}$ (n ungerade).

Miller-Rabin Zertifikat

$$\mathbb{Z}_n := \{0, 1, \ldots n-1\}$$

Für n prim, ist $(\mathbb{Z}_n, +, \cdot)$, '$\cdot$' und '$+$' mod n, ein Körper (mit 0 additives, und 1 multiplikatives neutrales Element).

In einem Körper hat die Gleichung $x^2 = 1$, zwei Lösungen:

$$x = 1 \text{ und } x = -1$$

Miller-Rabin Zertifikat

$$\mathbb{Z}_n := \{0, 1, \ldots n-1\}$$

Für n prim, ist $(\mathbb{Z}_n, +, \cdot)$, '$\cdot$' und '$+$' mod n, ein Körper (mit 0 additives, und 1 multiplikatives neutrales Element).

In einem Körper hat die Gleichung $x^2 = 1$, zwei Lösungen:

$$x = 1 \text{ und } x = -1 = n-1$$

Ist $n > 2$ prim und $a \in [n-1]$, dann gilt

- $a^{n-1} = 1$, (kleiner fermatscher Satz)

Miller-Rabin Zertifikat

$$\mathbb{Z}_n := \{0, 1, \ldots n-1\}$$

Für n prim, ist $(\mathbb{Z}_n, +, \cdot)$, '$\cdot$' und '$+$' mod n, ein Körper (mit 0 additives, und 1 multiplikatives neutrales Element).

In einem Körper hat die Gleichung $x^2 = 1$, zwei Lösungen:

$$x = 1 \text{ und } x = -1 = n-1$$

Ist $n > 2$ prim und $a \in [n-1]$, dann gilt

- $a^{n-1} = 1$, (kleiner fermatscher Satz)
- $a^{\frac{n-1}{2}} \in \{1, n-1\}$, (Körpereigenschaft oben)

Miller-Rabin Zertifikat

$$\mathbb{Z}_n := \{0, 1, \ldots n-1\}$$

Für n prim, ist $(\mathbb{Z}_n, +, \cdot)$, '$\cdot$' und '$+$' mod n, ein Körper (mit 0 additives, und 1 multiplikatives neutrales Element).

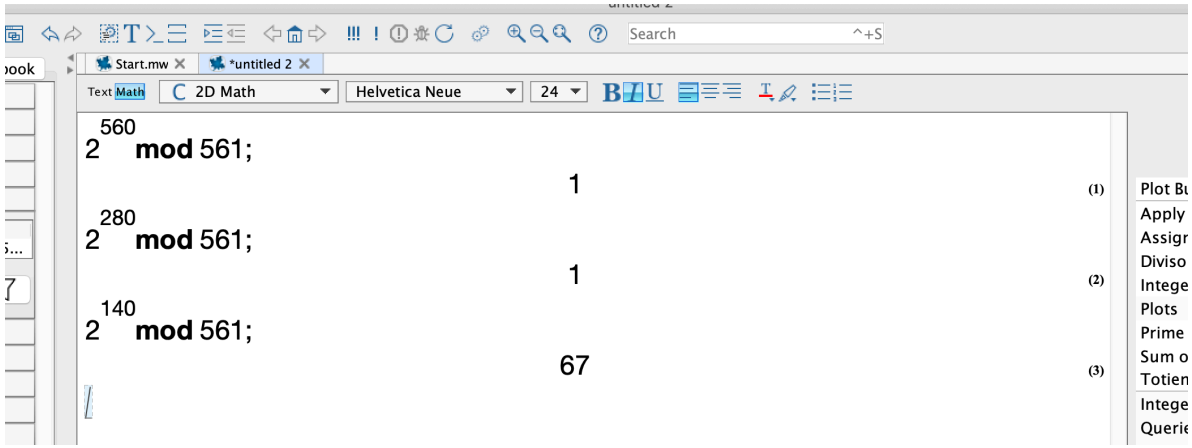
In einem Körper hat die Gleichung $x^2 = 1$, zwei Lösungen:

$$x = 1 \text{ und } x = -1 = n-1$$

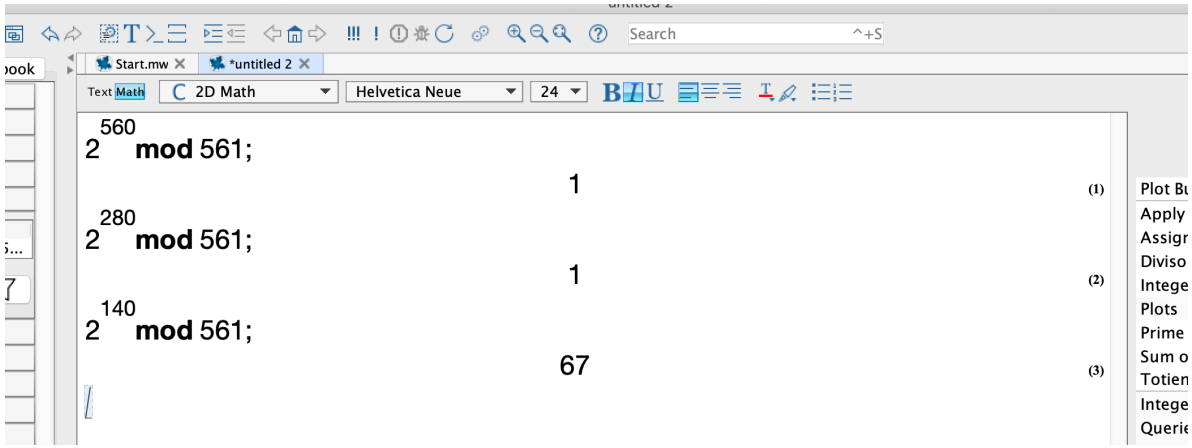
Ist $n > 2$ prim und $a \in [n-1]$, dann gilt

- $a^{n-1} = 1$, (kleiner fermatscher Satz)
- $a^{\frac{n-1}{2}} \in \{1, n-1\}$, (Körpereigenschaft oben)
- falls $a^{\frac{n-1}{2}} = 1$ und $\frac{n-1}{2}$ gerade, dann $a^{\frac{n-1}{4}} \in \{1, n-1\}$,
- ... bis $a^{\frac{n-1}{2^i}} = n-1$ oder $\frac{n-1}{2^i}$ ungerade.

Miller-Rabin Zertifikat



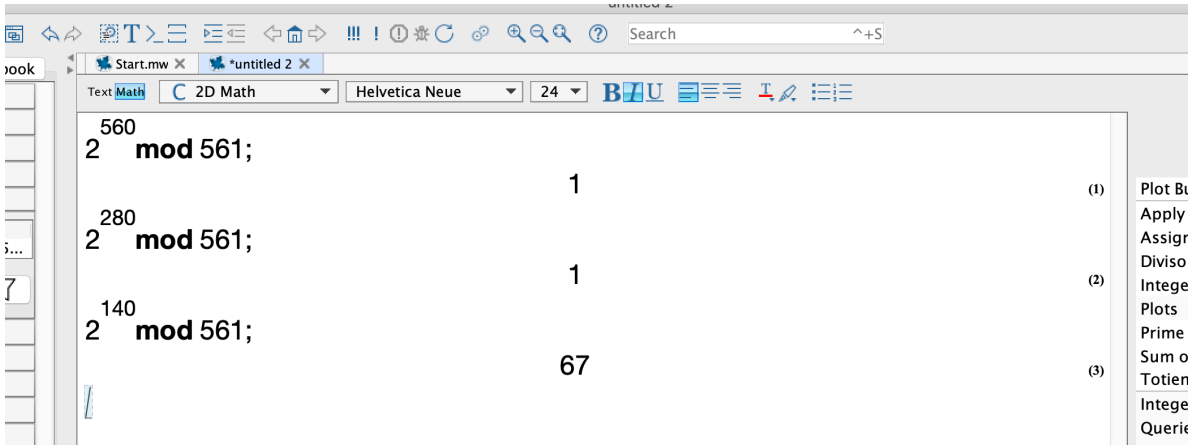
Miller-Rabin Zertifikat



Für $2 < n \in \mathbb{N}$, sei $n - 1 = 2^k d$, mit $d \in \mathbb{N}$ ungerade und $k \in \mathbb{N}_0$.
$a \in [n-1]$ is Miller-Rabin Zertifikat für "n nicht prim", falls

$$(a^d, a^{2d}, \ldots, a^{2^k d}) \neq \begin{cases} (1, 1, 1, \ldots, 1) & (\Leftrightarrow a^d \bmod n = 1) \\ (*, \ldots, *, n-1, 1, \ldots, 1) & (\Leftrightarrow \exists i : a^{2^i d} \bmod n = n - 1) \end{cases}$$

Miller-Rabin Zertifikat



Für $2 < n \in \mathbb{N}$, sei $n - 1 = 2^k d$, mit $d \in \mathbb{N}$ ungerade und $k \in \mathbb{N}_0$.
$a \in [n-1]$ is Miller-Rabin Zertifikat für "n nicht prim", falls

$$(a^d, a^{2d}, \ldots, a^{2^k d}) \neq \begin{cases} (1, 1, 1, \ldots, 1) & (\Leftrightarrow a^d \bmod n = 1) \\ (*, \ldots, *, n-1, 1, \ldots, 1) & (\Leftrightarrow \exists i : a^{2^i d} \bmod n = n-1) \end{cases}$$

Für $n < 1\,373\,653$, ist immer ein $a \in \{2, 3\}$ Miller-Rabin Zertifikat.

Miller-Rabin Test

Miller-Rabin-Primzahltest(n) ($n > 1$, ungerade!)

1: $d, k \in \mathbb{N}$, mit $n - 1 = 2^k d$, d ungerade
2: Wähle $a \in [n-1]$, zufällig gleichverteilt
3: **if** $a^d \bmod n \neq 1$ und $\nexists i < k : a^{2^i d} \bmod n = n - 1$ **then**
4: **return** 'keine Primzahl'
5: **else**
6: **return** 'Primzahl'

- Die Ausgabe 'keine Primzahl' ist immer richtig.
- Die Ausgabe 'Primzahl' ist falsch mit W'keit $\leq \frac{1}{4}$.

Anmerkungen

- Neben dem Miller-Rabin Test, der in der Praxis verwendet wird, gibt es auch den Solovay-Strassen Test, beides erste Klassiker für randomisierte Algorithmen aus den 70er Jahren.
- Es gibt einen deterministischen polynomiellen Primzahltest, der AKS Primzahltest [Agrawal-Kayal-Saxena 2002].
- Obwohl man schnell feststellen kann, ob eine Zahl nicht prim ist, d.h. einen nichtrivialen Teiler hat, ist es nicht bekannt, wie man einen solchen Teiler effizient findet (darauf basiert die RSA Verschlüsselung).
- Der Test "$\mathrm{ggT}(a, n) > 1$" im Fermat-Primzahltest ist redundant, weil $\mathrm{ggT}(a, n) > 1 \Rightarrow a^{n-1} \not\equiv 1 \pmod{n}$ gezeigt werden kann.

Lange Pfade

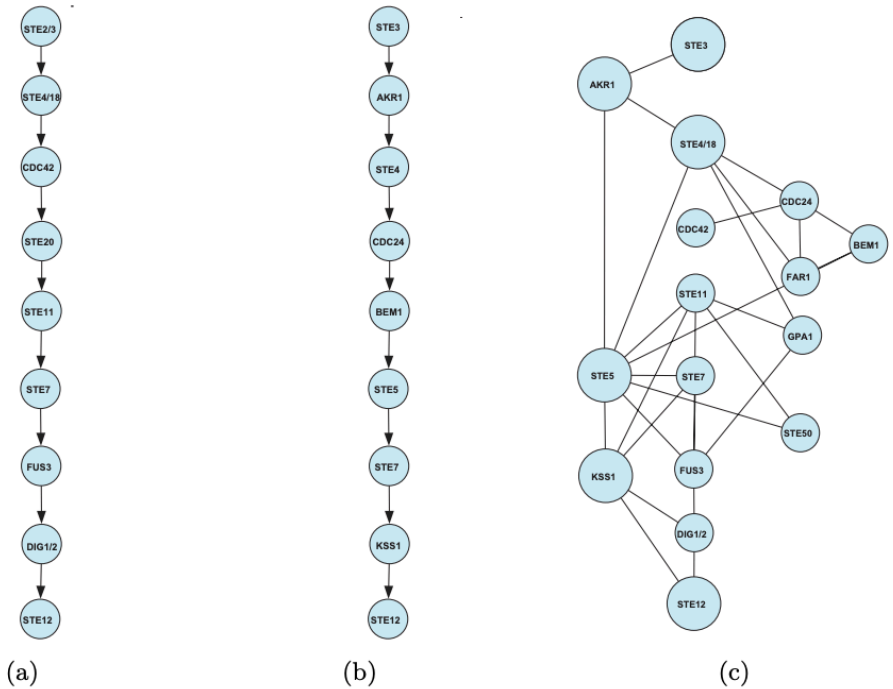


Fig. 2. The pheromone response signaling pathway in yeast. (a) The main chain of the known pathway, adapted from [13]. (b) The best path of the same length (9) in the network. (c) The assembly of all light-weight paths starting at STE3 and ending at STE12 that were identified in the network. Nodes that occur in at least half of the paths are drawn larger than the rest. Nodes that occur in less than 10% of the paths are omitted.

Abb. aus [Scott,Ideker,Karp,Sharan'05]

Problemstellung

LONG-PATH Problem. Gegeben (G, B), G ein Graph und $B \in \mathbb{N}_0$, stelle fest ob es einen Pfad der Länge B in G gibt.

Problemstellung

LONG-PATH Problem. Gegeben (G, B), G ein Graph und $B \in \mathbb{N}_0$, stelle fest ob es einen Pfad der Länge B in G gibt.

Ein **Pfad der Länge** ℓ in einem Graph $G = (V, E)$ ist eine Folge von paarweise verschiedenen Knoten

$$\langle v_0, v_1, \ldots, v_\ell \rangle, \text{ mit } \{v_{i-1}, v_i\} \in E \quad \text{für} \quad i = 1, \ldots \ell.$$

Es liegen $\ell + 1$ Knoten auf einem Pfad der Länge ℓ.

Das Problem ist (vermutlich) schwer

Theorie der $\mathcal{NP}$-Vollständigkeit lässt vermuten, dass es keinen Polynomialzeit-Algorithmus gibt, der entscheidet, ob in einem gegebenen Graph ein Hamiltonkreis existiert.

Man kann zeigen: Wenn es einen Polynomialzeit-Algorithmus für LONG-PATH gibt, dann gibt es einen Polynomialzeit-Algorithmus für das Hamiltonkreis Problem.

$\Rightarrow$ Wir vermuten, dass LONG-PATH schwer ist und keinen Polynomialzeit-Algorithmus hat.

Was gibt es also noch zu tun?

Kurze lange Pfade

In einer biologischen Anwendung geht es genau darum, in einem Graph lange Pfade zu finden. Diese sind aber in der Regel nur kurz. Also, was passiert, wenn B klein ist?

Konkret, wenn $B = O(\log n)$.

Wir brauchen

- $[n] := \{1, 2, \ldots, n\}$; $[n]^k$ ist die Menge der Folgen über $[n]$ der Länge k, es gilt $\left|[n]^k\right| = n^k$; $\binom{[n]}{k}$ ist die Menge der k-elementigen Teilmengen von $[n]$ und es gilt $\left|\binom{[n]}{k}\right| = \binom{n}{k}$.
- Für jeden Graph $G = (V, E)$ gilt $\sum_{v \in V} \deg(v) = 2|E|$.
- k Knoten kann man mit $[k]$ auf genau k^k Arten färben, $k!$ dieser Färbungen nutzen jede Farbe genau einmal.
- Für $c, n \in \mathbb{R}^+$, gilt $c^{\log n} = n^{\log c}$. Also, z.B. $2^{\log n} = n^{\log 2} = n$ und $2^{O(\log n)} = n^{O(1)}$ ist polynomiell in n.

Wir brauchen

- Für $n \in \mathbb{N}_0$ gilt $\sum_{i=0}^{n} \binom{n}{i} = 2^n$, eine Anwendung des Binomialsatzes: $\sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i} = (x+y)^n$.
- Für $n \in \mathbb{N}_0$ gilt $\frac{n!}{n^n} \geq e^{-n}$. Potenzreihenentwicklung der Exponentialfunktion:

$$e^n = \sum_{i=0}^{\infty} \frac{n^i}{i!} \geq \frac{n^n}{n!} \quad \text{(der Term in der Summe für } i = n\text{)}$$

- Wiederholt man ein Experiment mit Erfolgswahrscheinlichkeit p solange bis man Erfolg hat, dann ist der Erwartungswert der Anzahl der Versuche $\frac{1}{p}$ (geometrische Verteilung $\mathrm{Geo}(p)$).
- Dynamische Programmierung, ...

Bunte Pfade

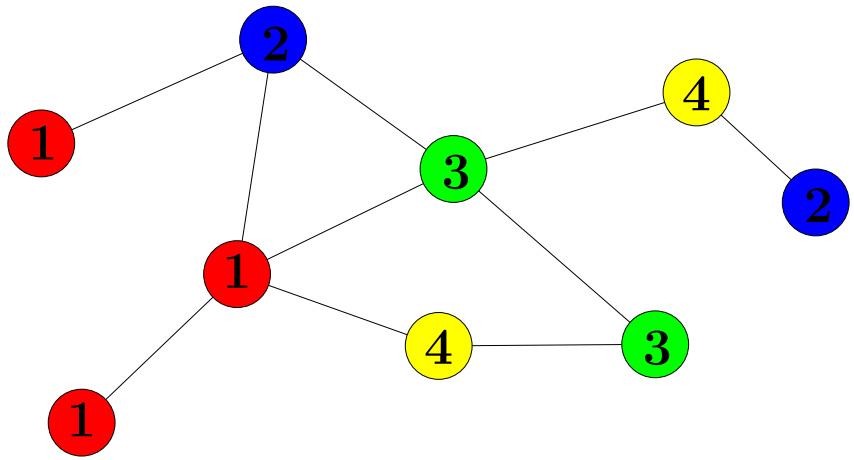
Wir ändern zwischenzeitlich das Problem.

$k \in \mathbb{N}$. Graph $G = (V, E)$ mit Färbung $\gamma \colon V \to [k]$ (nicht notwendigerweise gültige Färbung!).

Ein Pfad heisst **bunt**, falls alle seine Knoten verschiedene Farben haben.

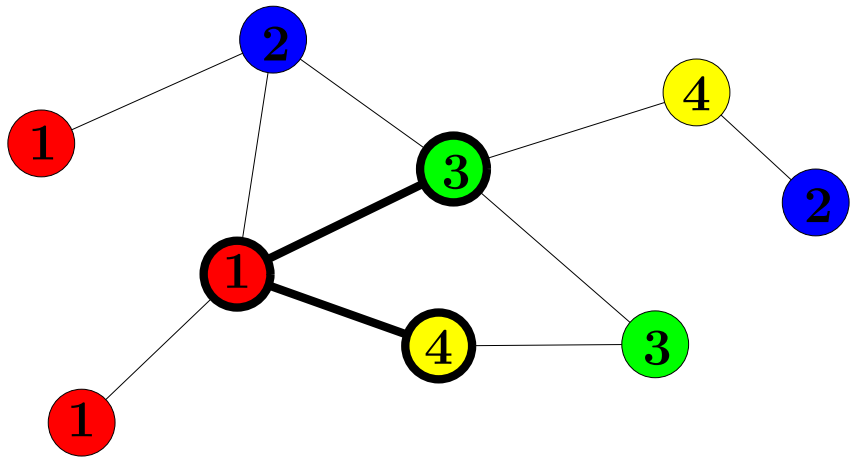
COLORFUL-PATH Problem. Gegeben (G, γ), $G = (V, E)$ ein Graph und $\gamma \colon V \to [k]$, stelle fest ob es einen bunten Pfad der Länge $k - 1$ (d.h. mit k Knoten) in G gibt.

Gefärbte Graphen, bunte Pfade



Eine Färbung $V \to [4]$.

Gefärbte Graphen, bunte Pfade



Ein bunter Pfad.

Farben auf bunten Pfaden zu Knoten v

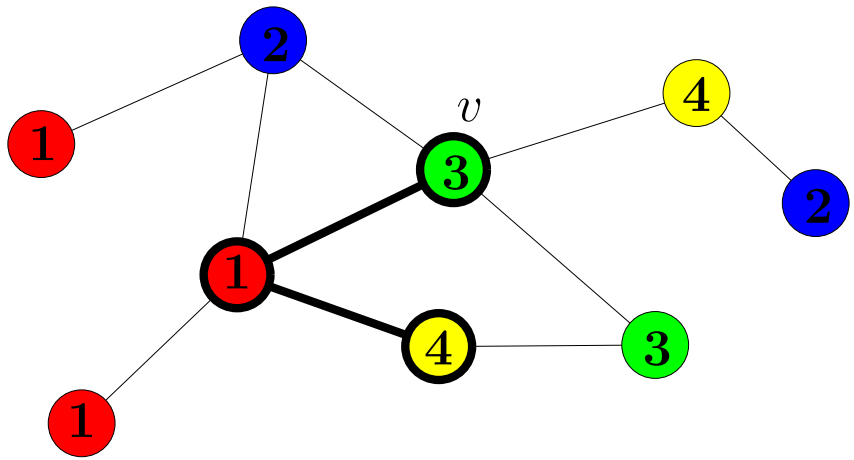
Wir fixieren einen Knoten v und und $i \in \mathbb{N}_0$. Wir betrachten nur bunte Pfade der Länge i (mit $i+1$ Knoten), die in v enden, und sammeln alle Farbmengen, die auf solchen Pfaden auftreten können.

$$P_i(v) := \{S \in \binom{[k]}{i+1} \quad | \quad \exists \text{ in } v \text{ endender genau mit } S \text{ gefärbter bunter Pfad}\};$$

Farben auf bunten Pfaden zu Knoten v

Wir fixieren einen Knoten v und und $i \in \mathbb{N}_0$. Wir betrachten nur bunte Pfade der Länge i (mit $i+1$ Knoten), die in v enden, und sammeln alle Farbmengen, die auf solchen Pfaden auftreten können.

$$P_i(v) := \{S \in \binom{[k]}{i+1} \quad | \quad \exists \text{ in } v \text{ endender genau mit } S \text{ gefärbter bunter Pfad}\};$$

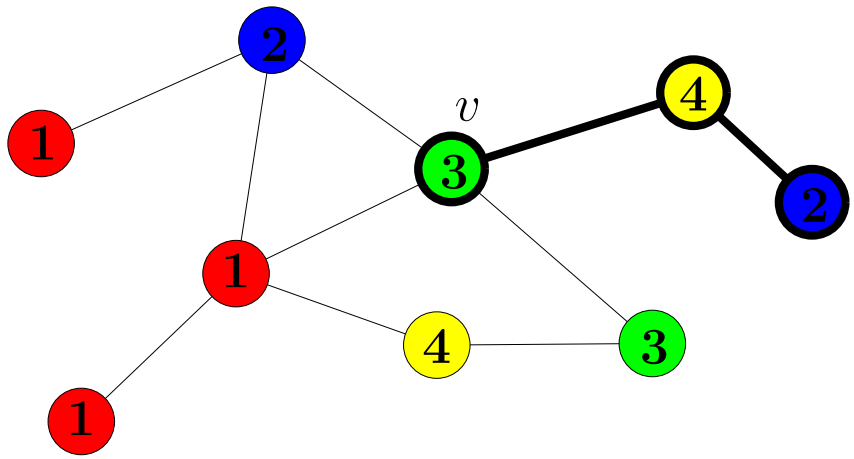


$$P_2(v) = \{\{4,1,3\}, \ldots\}$$

Farben auf bunten Pfaden zu Knoten v

Wir fixieren einen Knoten v und und $i \in \mathbb{N}_0$. Wir betrachten nur bunte Pfade der Länge i (mit $i+1$ Knoten), die in v enden, und sammeln alle Farbmengen, die auf solchen Pfaden auftreten können.

$$P_i(v) := \{S \in \binom{[k]}{i+1} \quad | \quad \exists \text{ in } v \text{ endender genau mit } S \text{ gefärbter bunter Pfad}\};$$



$$P_2(v) = \{\{4,1,3\},\{2,4,3\},\ldots\}$$

Farben auf bunten Pfaden zu Knoten v

Wir fixieren einen Knoten v und und $i \in \mathbb{N}_0$. Wir betrachten nur bunte Pfade der Länge i (mit $i+1$ Knoten), die in v enden, und sammeln alle Farbmengen, die auf solchen Pfaden auftreten können.

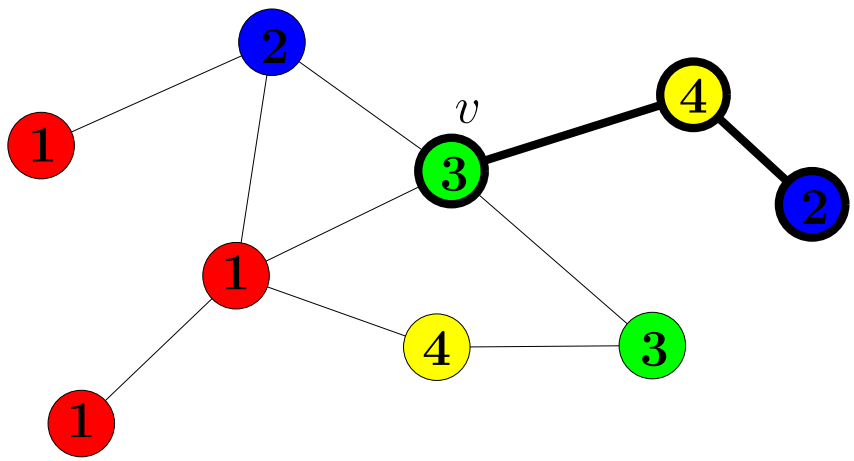
$$P_i(v) := \{S \in \binom{[k]}{i+1} \quad | \quad \exists \text{ in } v \text{ endender genau mit } S \text{ gefärbter bunter Pfad}\};$$

- $\forall S \in P_i(v) : \gamma(v) \in S$.
- $P_0(v) = \{\{\gamma(v)\}\}$.
- $P_1(v) = \{\{\gamma(x), \gamma(v)\} \mid x \in N(v), \gamma(x) \neq \gamma(v)\}$.
  $N(v) :=$ Menge der Nachbarn von v.
- $\exists$ bunter Pfad mit k Knoten $\Leftrightarrow \bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$.

Bunte Pfade bauen

Ein bunter Pfad der Länge i zu v besteht aus einem $\gamma(v)$-freien bunten Pfad der Länge $i-1$ zu einem Nachbarn x von v plus dem Schritt zu v.

$$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ und } \gamma(v) \notin R\}$$



Bunte Pfade bauen

Ein bunter Pfad der Länge i zu v besteht aus einem $\gamma(v)$-freien bunten Pfad der Länge $i-1$ zu einem Nachbarn x von v plus dem Schritt zu v.

$$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ und } \gamma(v) \notin R\}$$

Ähnlich zu $O(n^2 2^n)$ Algorithmus für Hamiltonkreise mit dynamischer Programmierung früher in der Vorlesung.

Bunte Pfade bauen

Ein bunter Pfad der Länge i zu v besteht aus einem $\gamma(v)$-freien bunten Pfad der Länge $i-1$ zu einem Nachbarn x von v plus dem Schritt zu v.

$$P_i(v) = \bigcup_{x \in N(v)} \{R \cup \{\gamma(v)\} \mid R \in P_{i-1}(x) \text{ und } \gamma(v) \notin R\}$$

Berechnung aller $P_i(v)$, $v \in V$, mit den $P_{i-1}(v)$, $v \in V$, gegeben:

Bunt(G, i) — G ein γ-gefärbter Graph

```
1: for all v ∈ V do
2:     P_i(v) ← ∅
3:     for all x ∈ N(v) do
4:         for all R ∈ P_{i−1}(x) mit γ(v) ∉ R do
5:             P_i(v) ← P_i(v) ∪ {R ∪ {γ(v)}}
```

Algorithmus

Bunt(G, i) — G ein γ-gefärbter Graph

```
1: for all v ∈ V do
2:     P_i(v) ← ∅
3:     for all x ∈ N(v) do
4:         for all R ∈ P_{i−1}(x) mit γ(v) ∉ R do
5:             P_i(v) ← P_i(v) ∪ {R ∪ {γ(v)}}
```

Algorithmus

Bunt(G, i) — G ein γ-gefärbter Graph

1: **for all** $v \in V$ **do**
2: $\quad P_i(v) \leftarrow \emptyset$
3: $\quad$ **for all** $x \in N(v)$ **do**
4: $\quad\quad$ **for all** $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ **do**
5: $\quad\quad\quad P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$

Entscheidung, ob es einen bunten Pfad mit k Knoten gibt:

Regenbogen(G, γ) — G Graph, γ k-Färbung

1: **for all** $v \in V$ **do** $P_0(v) \leftarrow \{\{\gamma(v)\}\}$
2: **for** $i = 1..k-1$ **do** Bunt(G, i)
3: **return** $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$

Analyse

Bunt(G, i)

1:	**for all** $v \in V$ **do**	
2:	$P_i(v) \leftarrow \emptyset$	
3:	**for all** $x \in N(v)$ **do**	$\deg(v)$
4:	**for all** $R \in P_{i-1}(x)$ mit $\gamma(v) \notin R$ **do**	$\lvert P_{i-1}(x)\rvert \cdot i$
5:	$P_i(v) \leftarrow P_i(v) \cup \{R \cup \{\gamma(v)\}\}$	

Mit $m := |E|$ und wegen $|P_{i-1}(x)| \leq \binom{k}{i}$ ist der Zeitaufwand

$$O\left(\overbrace{\sum_{v \in V} \deg(v)}^{2m} \cdot \binom{k}{i} \cdot i\right) = O\left(\binom{k}{i} \cdot i \cdot m\right) \ .$$

Analyse

Regenbogen(G, γ) — G Graph, γ k-Färbung

1: **for all** $v \in V$ **do** $P_0(v) \leftarrow \{\{\gamma(v)\}\}$
2: **for** $i = 1..k-1$ **do** Bunt(G, i)
3: **return** $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$

$$O\left(|V| + \sum_{i=1}^{k-1} \left(\binom{k}{i} \cdot i \cdot m\right) + |V|\right) = O(2^k km) \ .$$

wegen $\sum_{i=1}^{k-1} \binom{k}{i} \leq \sum_{i=0}^{k} \binom{k}{i} = 2^k$

Analyse

Regenbogen(G, γ) — G Graph, γ k-Färbung

1: **for all** $v \in V$ **do** $P_0(v) \leftarrow \{\{\gamma(v)\}\}$
2: **for** $i = 1..k-1$ **do** Bunt(G, i)
3: **return** $\bigcup_{v \in V} P_{k-1}(v) \neq \emptyset$

$$O\left(|V| + \sum_{i=1}^{k-1} \left(\binom{k}{i} \cdot i \cdot m\right) + |V|\right) = O(2^k km) \ .$$

wegen $\sum_{i=1}^{k-1} \binom{k}{i} \leq \sum_{i=0}^{k} \binom{k}{i} = 2^k$

D.h. für $k \leq \log n$: Laufzeit $O(mn \log n)$,
$k = O(\log n)$: Laufzeit $O(\text{poly}(n))$.

Auf gut Glück – Zufallsfärbungen

Ursprüngliches **LONG-PATH Problem**: Gegeben (G, B), G Graph, $B \in \mathbb{N}_0$, stelle fest ob es einen Pfad der Länge B in G gibt.

Setze $k := B + 1$, färbe G zufällig mit k Farben, und suche einen bunten Pfad mit k Knoten.

Auf gut Glück – Zufallsfärbungen

Ursprüngliches **LONG-PATH Problem**: Gegeben (G, B), G Graph, $B \in \mathbb{N}_0$, stelle fest ob es einen Pfad der Länge B in G gibt.

Setze $k := B + 1$, färbe G zufällig mit k Farben, und suche einen bunten Pfad mit k Knoten.

Angenommen G hat einen Pfad P mit k Knoten. Dann ist die W'keit p_{Erfolg}, dass es in (G, γ) einen bunten Pfad gibt

$$p_{\text{Erfolg}} := \Pr[\exists \text{ bunter Pfad mit } k \text{ Knoten}]$$
$$\geq \Pr[P \text{ ist bunt}] = \frac{k!}{k^k} \geq e^{-k} .$$

Erwartete Anzahl der Versuche, bis man einen bunten Pfad (und damit einen Pfad mit k Knoten findet): $\leq e^k$ $\qquad (\mathrm{Geo}(e^{-k}))$.

Viele zufällige Färbungen

Angenommen G hat einen Pfad mit k Knoten.

Ein Versuch:

- Laufzeit $O(2^k km)$. $\quad p_{\text{Erfolg}} \geq e^{-k}$.

$\lceil \lambda e^k \rceil$ Versuche:

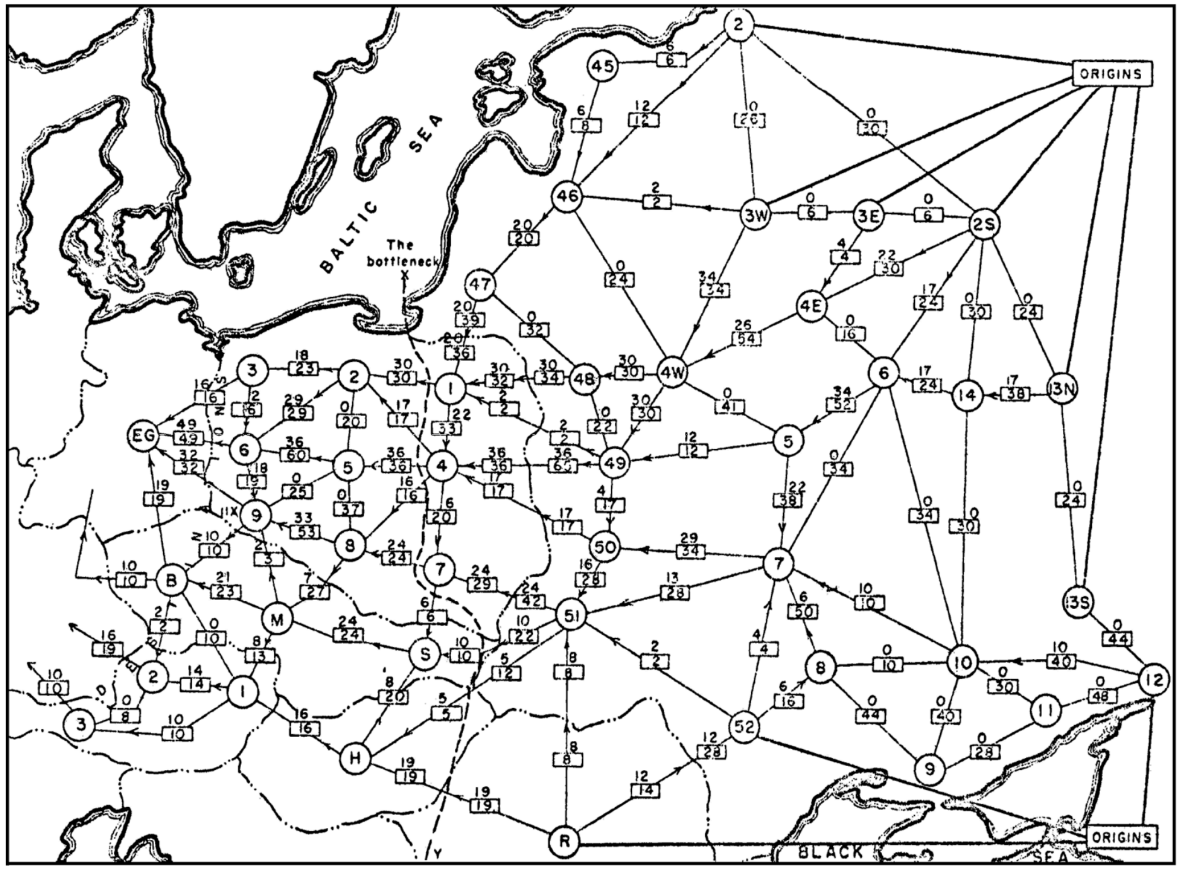
- Laufzeit $O(\lambda (2e)^k km)$.
- W'keit, dass der Algorithmus den Pfad nicht findet ist

$$\leq \left(1 - e^{-k}\right)^{\lceil \lambda e^k \rceil} \leq \left(e^{-e^{-k}}\right)^{\lceil \lambda e^k \rceil} \leq e^{-\lambda}.$$

Anmerkungen

- Die hier (nebst dynamischem Programmieren) verwendete Methode heisst **Color-Coding**. [Alon,Yuster, Zwick '95]
- $\exists$ randomisierte Verbesserung, die in Zeit $O(2^k \text{poly}(n))$ läuft (statt $O((2e)^k \text{poly}(n))$. [Williams '09]
- $\exists$ deterministische Polynomialzeit-Variante für $B = O(\log n)$.
- Will man einen Pfad der Länge B tatsächlich finden (statt nur die Existenz festzustellen), kann man den Algorithmus leicht adaptieren. Man merkt sich dazu einfach zu jedem $S \in P_i(v)$ einen genau mit S gefärbten bunten Pfad $\langle u_0, u_1, \ldots, u_i \rangle$, $u_i = v$.
- Das Problem wird einfach in gerichteten azyklischen Graphen: Man gewichtet alle Kanten einfach mit -1 und berechnet kürzeste Pfade (siehe *Algorithmen und Datenstruktur*-Vorlesung im Herbst).

Flüsse in Netzwerken: Einführung & Modellierung



Ursprung: Interesse am russischen Schienennetz,
[A.N. Tolstoĭ '30] und [Harris&Ross '55] (diese Abb.).

Problemstellung

MaxFlow Problem. Gegeben ein Netzwerk, finde einen Fluss grössten Werts. (Netzwerk?, Fluss?, Wert?)

Problemstellung

MaxFlow Problem. Gegeben ein Netzwerk, finde einen Fluss grössten Werts. (Netzwerk?, Fluss?, Wert?)

Verkehrsflüsse
Geldflüsse
Transportprobleme

elektrische Leitungen
mit Widerständen
Bildsegmentierung
in der Bildverarbeitung

Problemstellung

MaxFlow Problem. Gegeben ein Netzwerk, finde einen Fluss grössten Werts. (Netzwerk?, Fluss?, Wert?)

Verkehrsflüsse
Geldflüsse
Transportprobleme

elektrische Leitungen
mit Widerständen
Bildsegmentierung
in der Bildverarbeitung

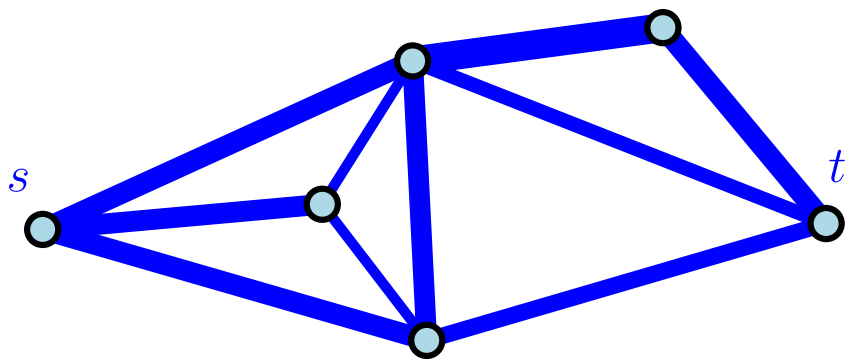
Matchings in Graphen
Schnitte (Cuts) in Graphen
Disjunkte Wege in Graphen

Grenzbereich polynomiell vs.
nicht polynomiell

Netzwerk – Definition

Ein **Netzwerk** ist ein Tupel $N = (V, A, c, s, t)$, wobei gilt:

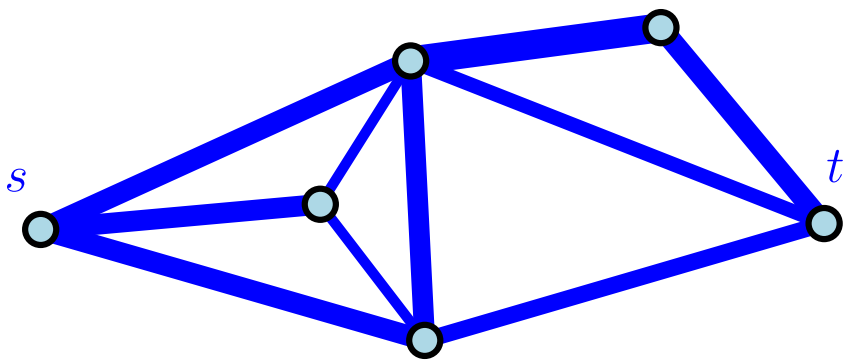
- (V, A) ist ein gerichteter Graph (ohne Schleifen),
- $s \in V$, die **Quelle**,
- $t \in V \setminus \{s\}$, die **Senke**, und
- $c : A \to \mathbb{R}_0^+$, die **Kapazitätsfunktion**.

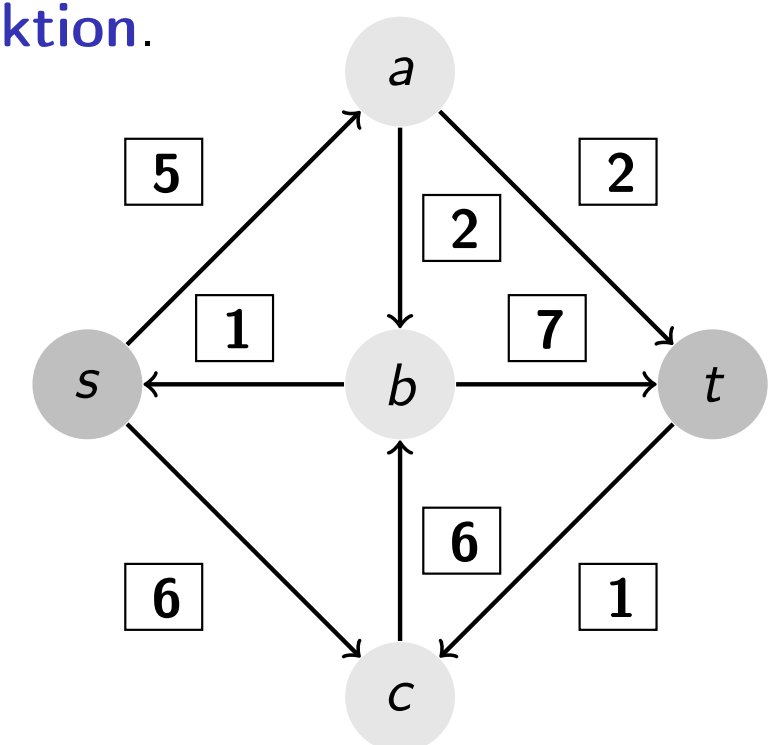


Netzwerk – Definition

Ein **Netzwerk** ist ein Tupel $N = (V, A, c, s, t)$, wobei gilt:

- (V, A) ist ein gerichteter Graph (ohne Schleifen),
- $s \in V$, die **Quelle**,
- $t \in V \setminus \{s\}$, die **Senke**, und
- $c : A \to \mathbb{R}_0^+$, die **Kapazitätsfunktion**.





Fluss – Definition

Sei $N = (V, A, c, s, t)$ ein Netzwerk. Ein **Fluss** in N ist eine Funktion $f : A \to \mathbb{R}$ mit den Bedingungen

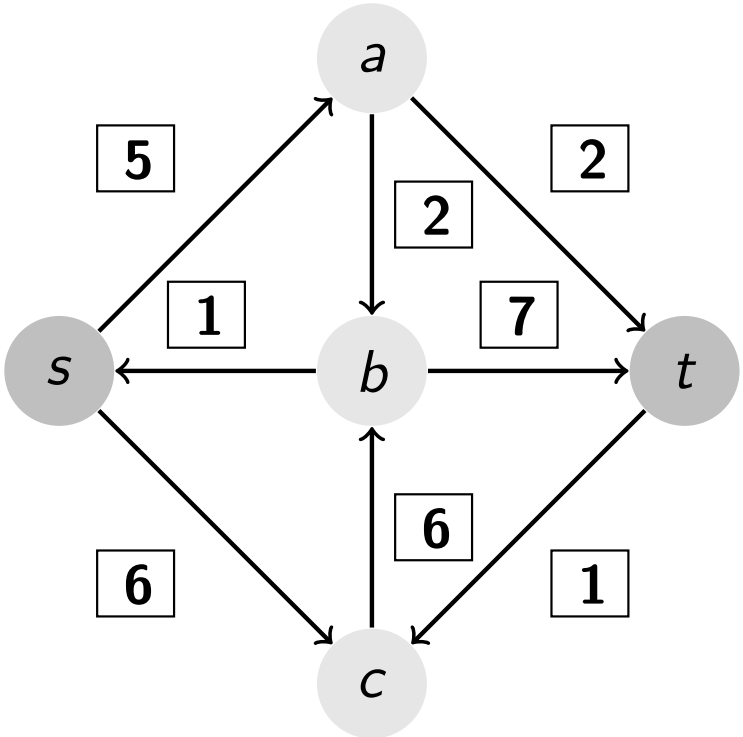
- **Zulässigkeit**: $0 \leq f(e) \leq c(e)$ für alle $e \in A$.
- **Flusserhaltung**: Für alle $v \in V \setminus \{s, t\}$ gilt

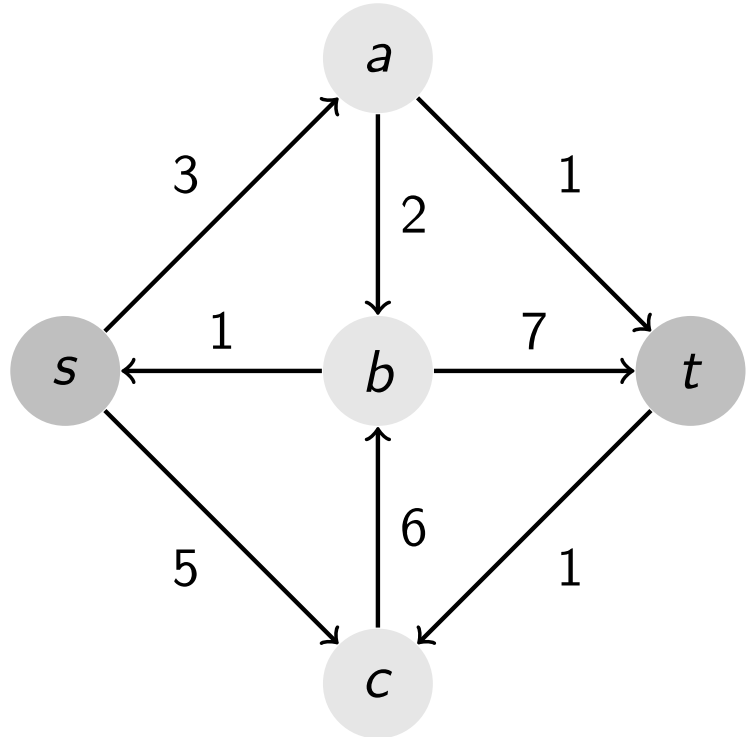
$$\sum_{u \in V : (u,v) \in A} f(u, v) = \sum_{u \in V : (v,u) \in A} f(v, u) \ .$$

Der **Wert** eines Flusses f ist definiert als

$$\text{val}(f) := \text{netoutflow}(s) := \sum_{u \in V : (s,u) \in A} f(s, u) - \sum_{u \in V : (u,s) \in A} f(u, s) \ .$$

Netzwerk und Fluss





Fluss mit Wert $3 - 1 + 5 = 7$

Nettozufluss der Senke

Intuition: Soviel, wie bei der Quelle herausfliesst, muss bei der Senke hineinfliessen.

Nettozufluss der Senke

Intuition: Soviel, wie bei der Quelle herausfliesst, muss bei der Senke hineinfliessen.

Lemma

Der **Nettozufluss** *der Senke t gleicht dem Wert des Flusses, d.h.*

$$\text{netinflow}(t) := \sum_{u \in V:\,(u,t)\in A} f(u,t) - \sum_{u \in V:\,(t,u)\in A} f(t,u) \quad = \quad \text{val}(f)\ .$$

Nettozufluss der Senke – Beweis

$$
\begin{aligned}
0 &= \sum_{(v,u)\in A} f(v,u) - \sum_{(u,v)\in A} f(u,v) \\
&= \sum_{v\in V} \underbrace{\left(\sum_{u\in V:\,(v,u)\in A} f(v,u) - \sum_{u\in V:\,(u,v)\in A} f(u,v) \right)}_{=0 \text{ für } v\notin\{s,t\}} \\
&= \underbrace{\left(\sum_{u\in V:\,(s,u)\in A} f(s,u) - \sum_{u\in V:\,(u,s)\in A} f(u,s) \right)}_{=\mathrm{val}(f)} \\
&\qquad + \underbrace{\left(\sum_{u\in V:\,(t,u)\in A} f(t,u) - \sum_{u\in V:\,(u,t)\in A} f(u,t) \right)}_{=-\mathrm{netinflow}(t)}
\end{aligned}
$$

Problemstellung

MaxFlow Problem. Gegeben ein Netzwerk $N = (V, A, c, s, t)$, finde einen Fluss f grössten Werts (einen **maximalen Fluss**).

- Gibt es immer einen solchen maximalen Fluss? Es könnte ein ähnliches Phänomen auftreten wie beim offenen Intervall $(0, 1)$: Es gibt keine grösste Zahl, kein Maximum.
- Wie erkennt man, dass ein Fluss maximal ist? Was ist ein „einfacher“ Beweis für die Maximalität eines Flusses?

Wir betrachten dazu vorerst Schnitte in Graphen.

Schnitt - Definition

Ein *s*-*t*-**Schnitt** für ein Netzwerk (V, A, c, s, t) ist eine Partition (S, T) von V mit $s \in S$ und $t \in T$. Die **Kapazität** eines s-t-Schnitts (S, T) ist durch

$$\text{cap}(S, T) := \sum_{(u,w) \in (S \times T) \cap A} c(u, w)$$

definiert. (Partition (S, T): $S \cup T = V$ und $S \cap T = \emptyset$)

Schnitt - Definition

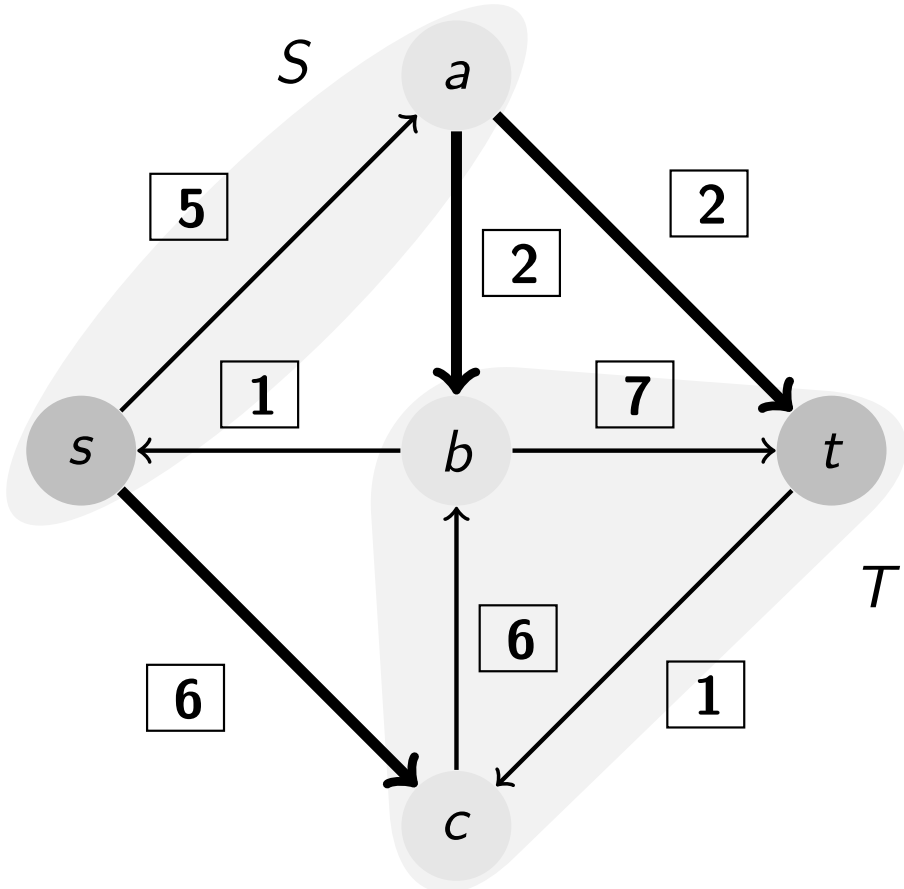
Ein *s*-*t*-**Schnitt** für ein Netzwerk (V, A, c, s, t) ist eine Partition (S, T) von V mit $s \in S$ und $t \in T$. Die **Kapazität** eines s-t-Schnitts (S, T) ist durch

$$\text{cap}(S, T) := \sum_{(u,w) \in (S \times T) \cap A} c(u, w)$$

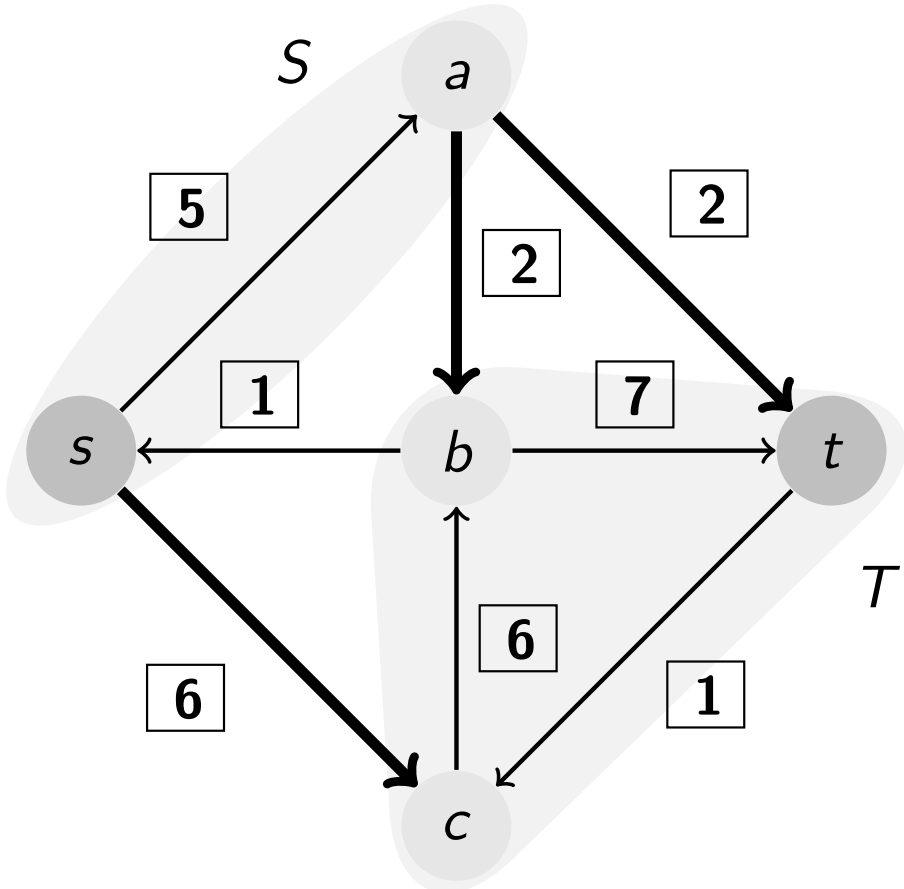
definiert. (Partition (S, T): $S \cup T = V$ und $S \cap T = \emptyset$)

Wichtig: Die Kapazität eines Schnitts (S, T) ignoriert die Kanten von T nach S!

Schnitt



Schnitt



Schnitt mit Kapazität $6 + 2 + 2 = 10$.

Schnitt vs. Fluss

Lemma

Ist f ein Fluss und (S, T) ein s-t-Schnitt in einem Netzwerk (V, A, c, s, t), so gilt

$$\mathsf{val}(f) \leq \mathsf{cap}(S, T) \ .$$

Schnitt vs. Fluss

Lemma
Ist f ein Fluss und (S, T) ein s-t-Schnitt in einem Netzwerk (V, A, c, s, t), so gilt

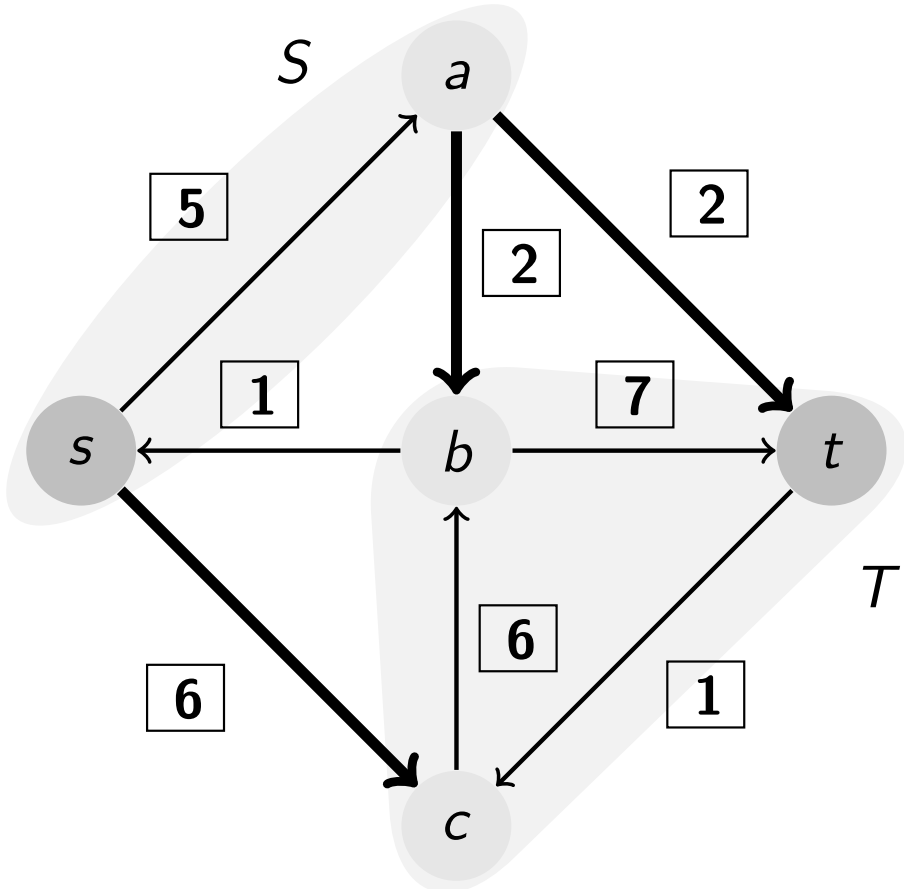
$$\mathrm{val}(f) \leq \mathrm{cap}(S, T) \ .$$

Ein Fluss kann nie grösser sein als die Kapazität eines s-t-Schnitts.

Finden wir zu einem Fluss f einen s-t-Schnitt (S, T) mit $\mathrm{cap}(S, T) = \mathrm{val}(f)$, so ist f ein maximaler Fluss.

Der Schnitt (S, T) is ein einfacher Beweis (ein einfaches Zertifikat) für die Maximalität von f.

Schnitt vs. Fluss



Schnitt mit Kapazität $6 + 2 + 2 = 10$.

Lemma „$\mathrm{val}(f) \leq \mathrm{cap}(S, T)$“ - Beweis

Für eine Partition (U, W) von V sei

$$f(U, W) := \sum_{(u,w) \in (U \times W) \cap A} f(u, w) \ .$$

Wir behaupten

$$\mathrm{val}(f) \overset{\text{(i)}}{=} f(S, T) - f(T, S) \overset{\text{(ii)}}{\leq} f(S, T) \overset{\text{(iii)}}{\leq} \mathrm{cap}(S, T),$$

(ii) Folgt aus Nichtnegativität des Flusses auf jeder Kante.
(iii) Folgt aus der Kapazitätsbeschränkung „$f(u, w) \leq c(u, w)$“.

(i) gilt für $S = \{s\}$ bei Definition, und für $T = \{t\}$ wg. $\mathrm{netinflow}(t) = \mathrm{val}(f)$. Allgemein ...

Lemma „val$(f) \leq \mathrm{cap}(S, T)$“ - Beweis

$$
\begin{aligned}
\mathrm{val}(f) &= \sum_{u \in V:\,(s,u) \in A} f(s,u) - \sum_{u \in V:\,(u,s) \in A} f(u,s) \\
&= \sum_{v \in S} \underbrace{\left(\sum_{u \in V:\,(v,u) \in A} f(v,u) - \sum_{u \in V:\,(u,v) \in A} f(u,v) \right)}_{=0 \quad \text{für} \quad v \neq s} \\
&= \sum_{(u,w) \in (S \times T) \cap A} f(u,w) - \sum_{(u,w) \in (T \times S) \cap A} f(u,w) \\
&= f(S,T) - f(T,S)
\end{aligned}
$$

Maxflow-Mincut

Ein einfaches Maximalitätszertifikat existiert immer.

Satz („**Maxflow-Mincut Theorem**“)

Jedes Netzwerk $N = (V, A, c, s, t)$ *erfüllt*

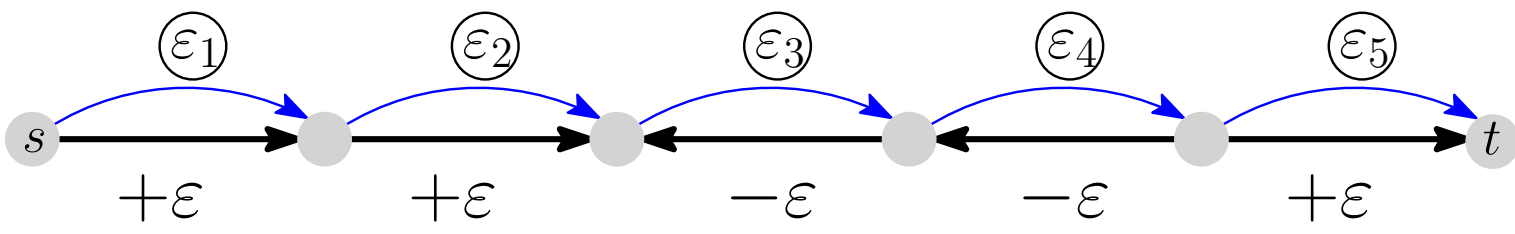
$$\max_{f \text{ Fluss}} \mathrm{val}(f) = \min_{(S,T) \text{ } s\text{-}t\text{-Schnitt}} \mathrm{cap}(S, T)$$

Beachte: Da es nur endlich viele s-t-Schnitte gibt, d.h.

- s-t-MinCut ist ein endliches algorithmisches Problem, und
- ein minimaler Schnitt existiert immer.

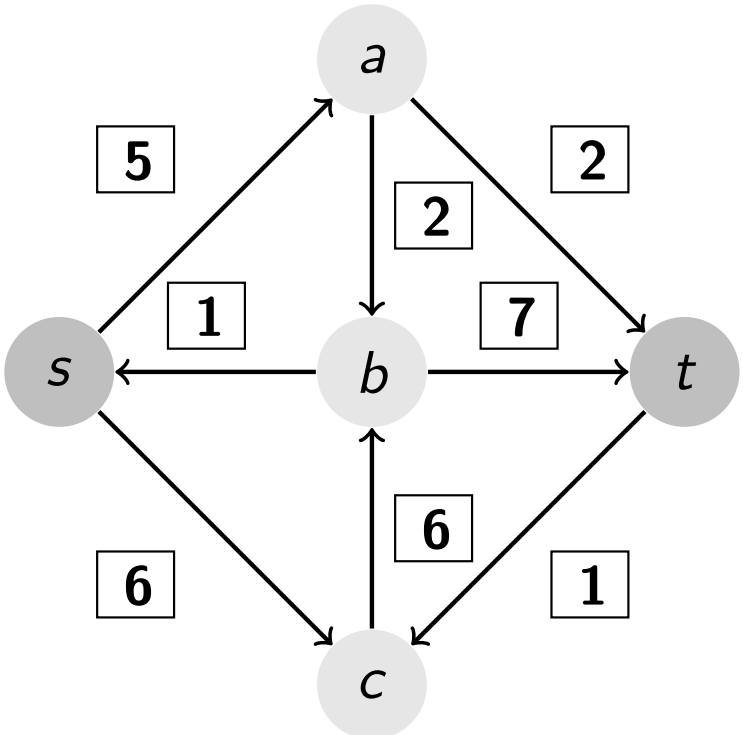
Am Ende werden wir doch s-t-MinCut mit MaxFlow lösen.

Flüsse in Netzwerken: Algorithmen

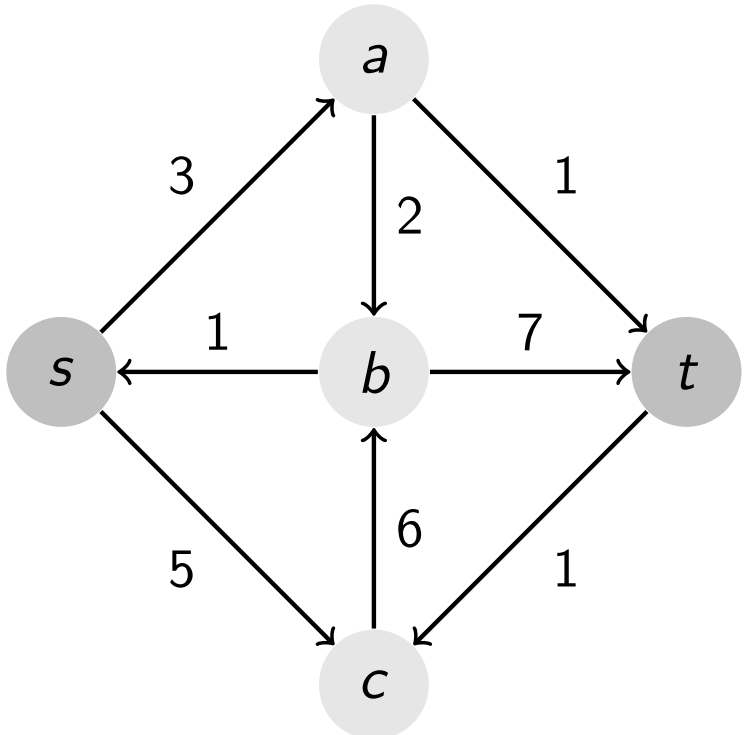


$\varepsilon := \min_i \varepsilon_i$

Netzwerke und Flüsse

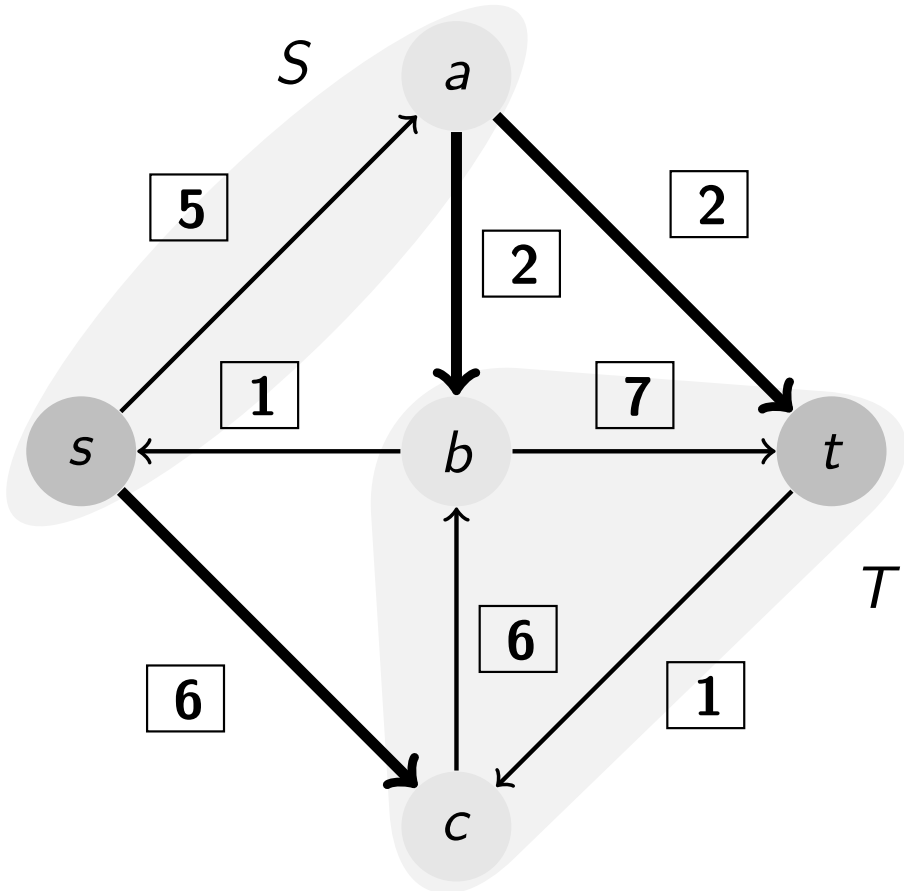


Netzwerk $N = (V, A, c, s, t)$.



Fluss f mit Wert $3 - 1 + 5 = 7$.

Schnitt



s-t-Schnitt (S, T) mit Kapazität $6 + 2 + 2 = 10$.

Schnitt vs. Fluss

Lemma

Ist f ein Fluss und (S, T) ein s-t-Schnitt in einem Netzwerk, so gilt

$$\mathrm{val}(f) \leq \mathrm{cap}(S, T) \ .$$

(bewiesen)

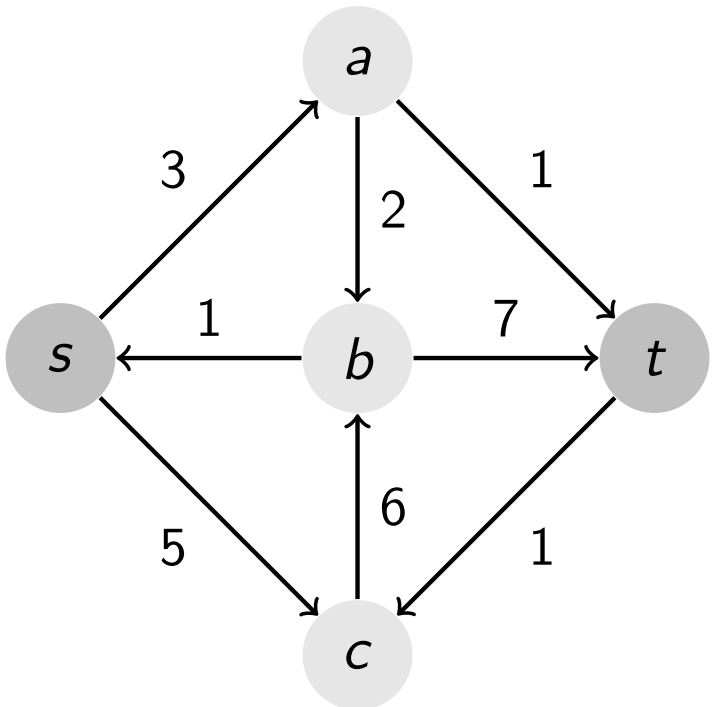
Schnitt vs. Fluss

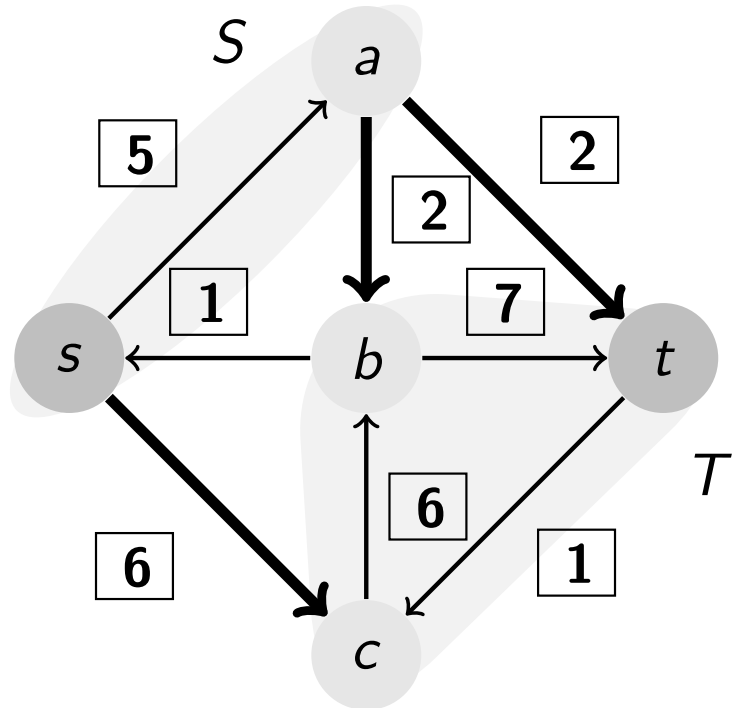
Lemma

Ist f ein Fluss und (S, T) ein s-t-Schnitt in einem Netzwerk, so gilt

$$\mathrm{val}(f) \leq \mathrm{cap}(S, T) \ .$$

(bewiesen)





Fluss mit Wert 7 $\boxed{7 \leq 10}$ Schnitt mit Kapaziät 10

Schnitt vs. Fluss

Lemma

Ist f ein Fluss und (S, T) ein s-t-Schnitt in einem Netzwerk, so gilt

$$\mathrm{val}(f) \leq \mathrm{cap}(S, T) \ .$$

(bewiesen)

Satz („Maxflow-Mincut Theorem“)

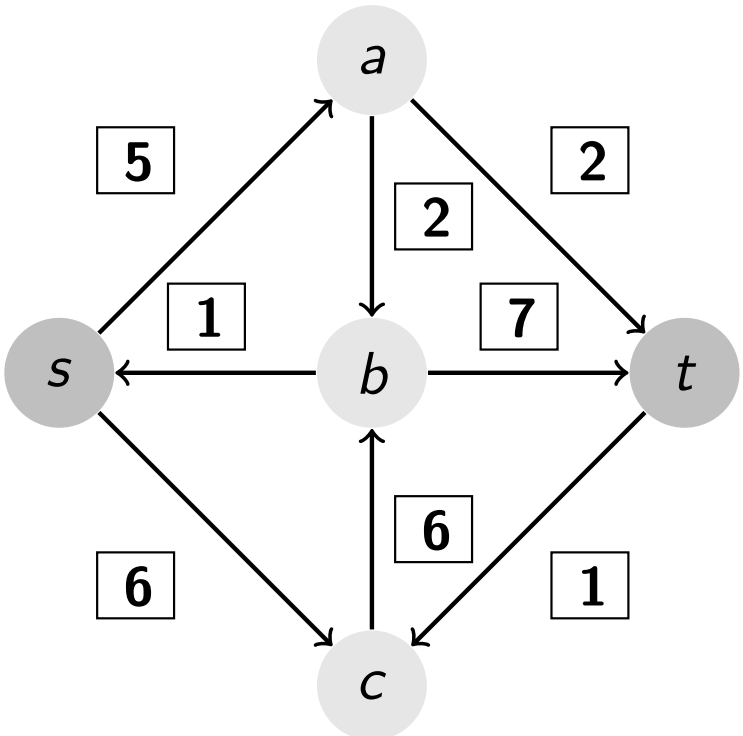
Jedes Netzwerk erfüllt

$$\max_{f \text{ Fluss}} \mathrm{val}(f) = \min_{(S,T) \text{ s-t-Schnitt}} \mathrm{cap}(S, T) \ .$$

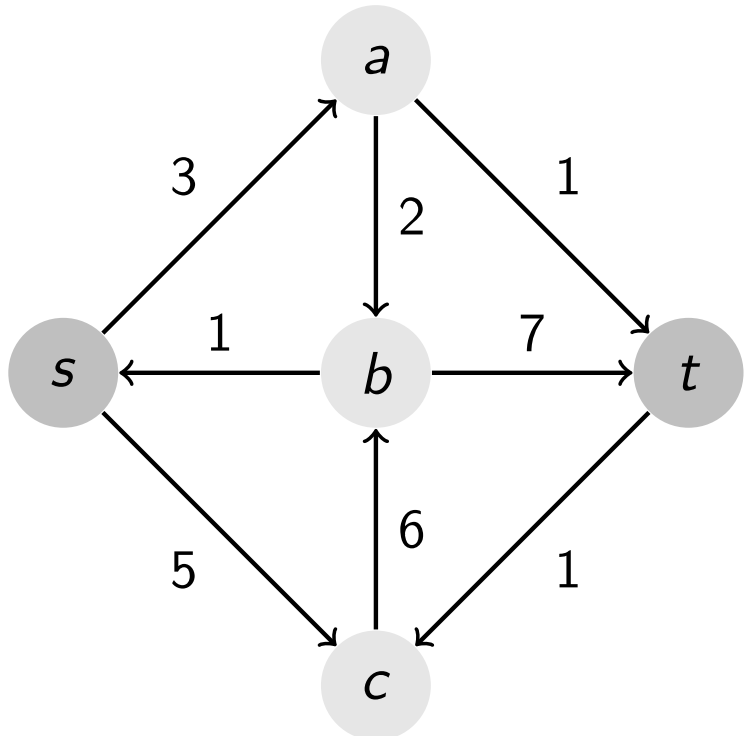
(noch nicht bewiesen)

Ziel: Algorithmus und Beweis des Maxflow-Mincut Theorem.

Verbessern eines gegebenen Flusses (1)

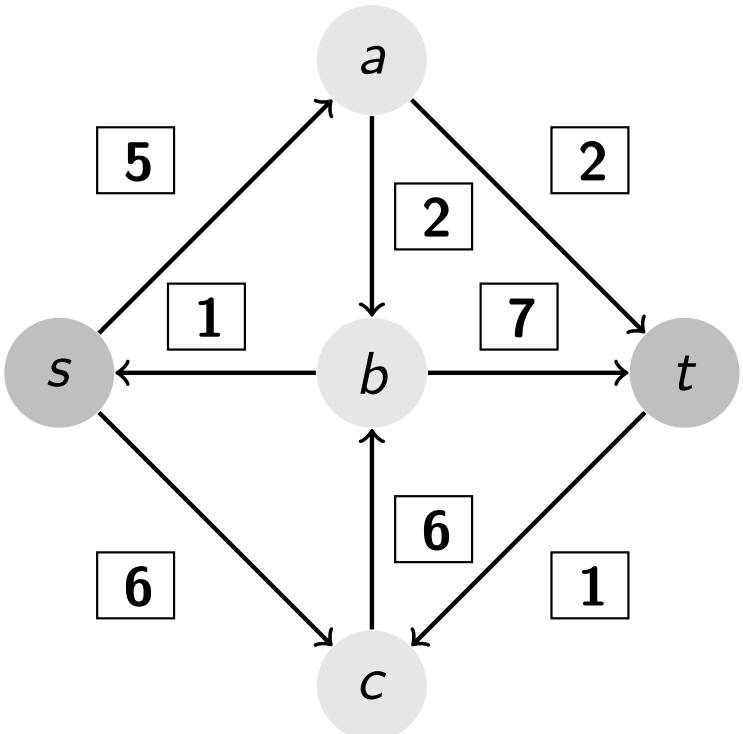


Netzwerk $N = (V, A, c, s, t)$

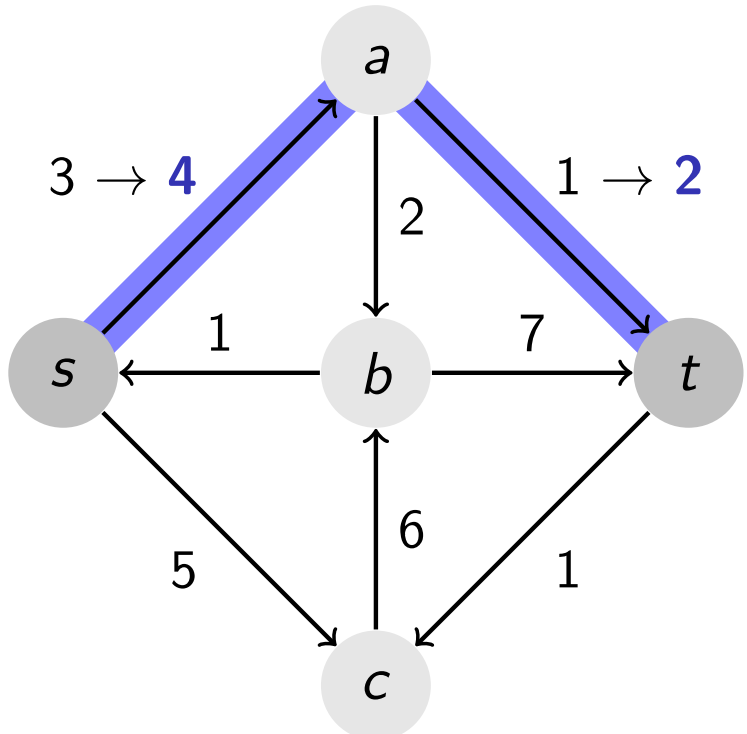


Fluss mit Wert $3 - 1 + 5 = 7$

Verbessern eines gegebenen Flusses (1)

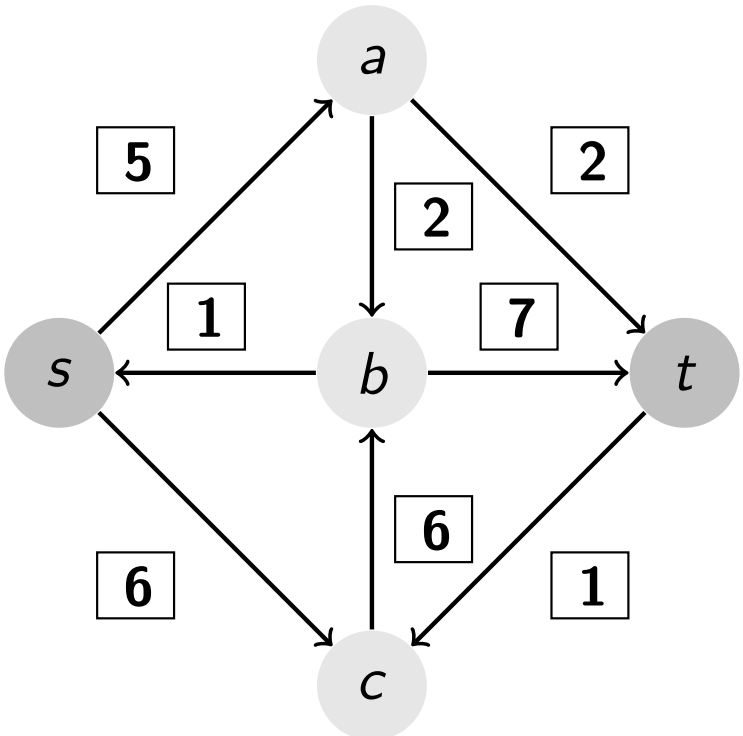


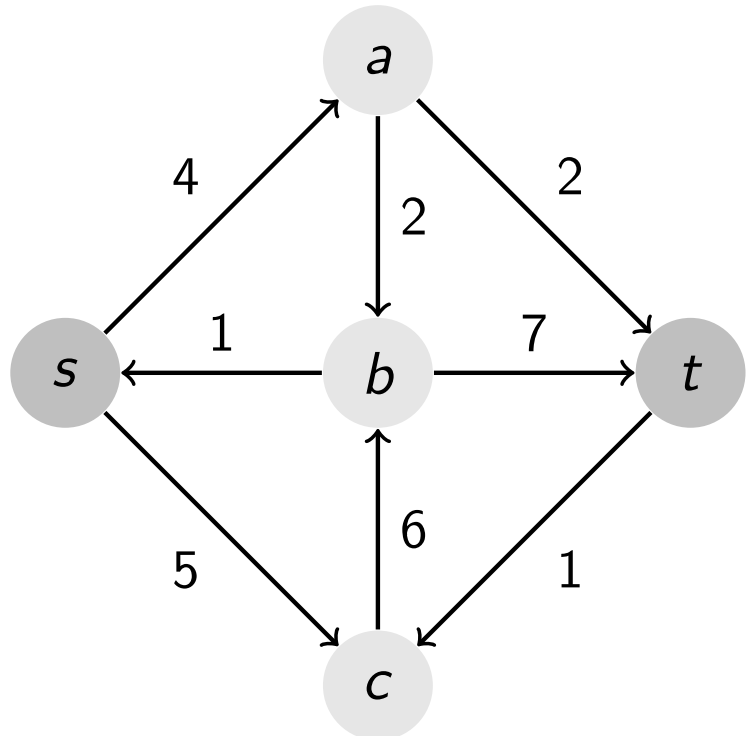
Netzwerk $N = (V, A, c, s, t)$



Fluss mit Wert $4 - 1 + 5 = 8$

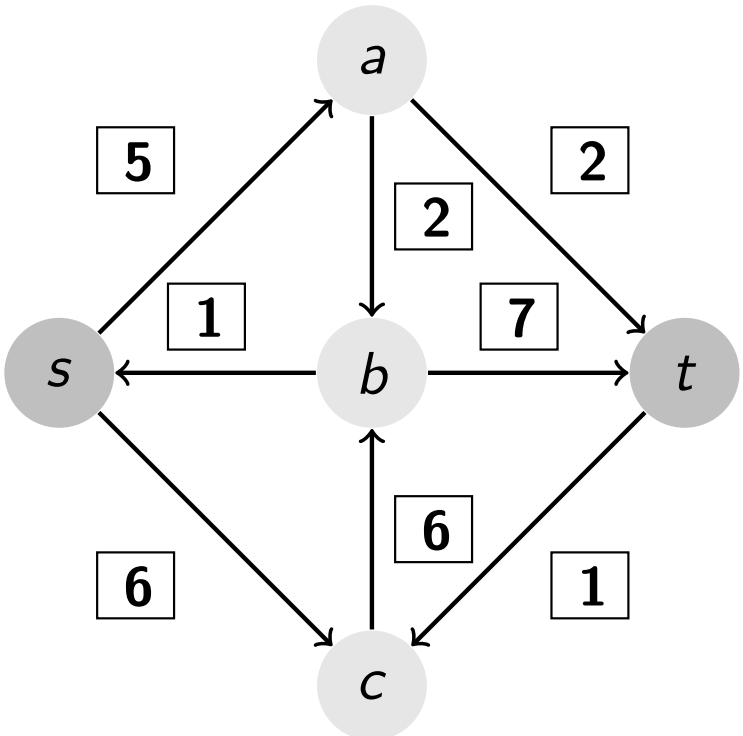
Verbessern eines gegebenen Flusses (2)

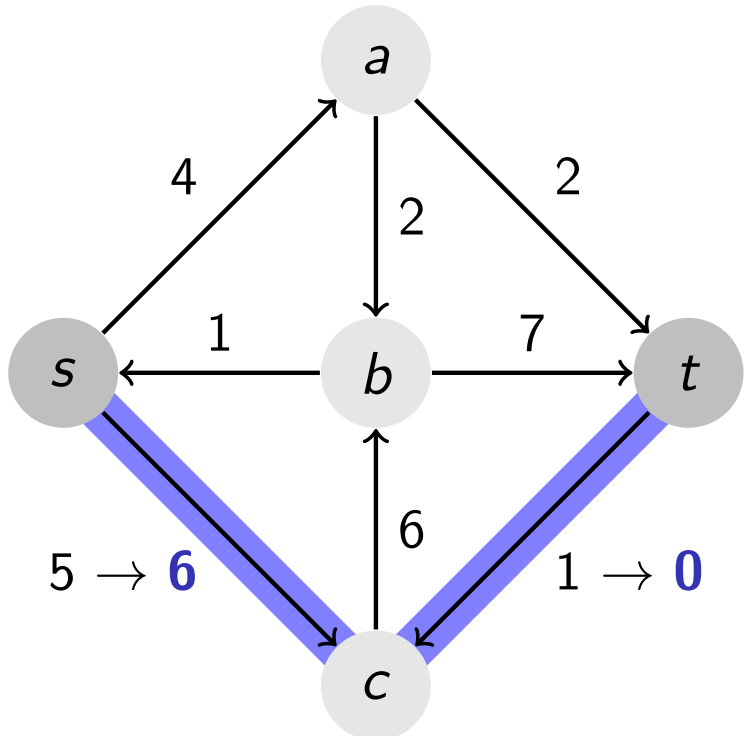




Fluss mit Wert $4 - 1 + 5 = 8$

Verbessern eines gegebenen Flusses (2)

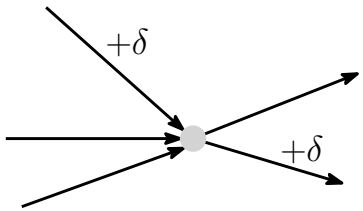




Fluss mit Wert $4 - 1 + \mathbf{6} = \mathbf{9}$

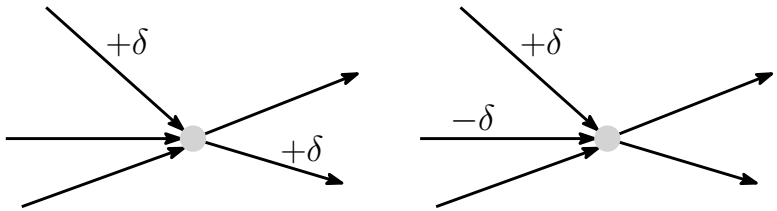
Flusserhaltungserhaltung

Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:



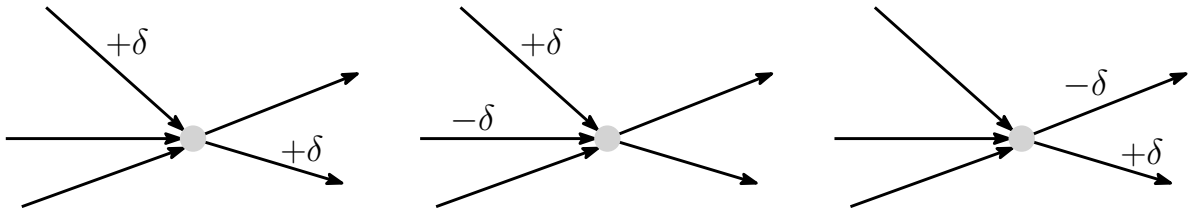
Flusserhaltungserhaltung

Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:



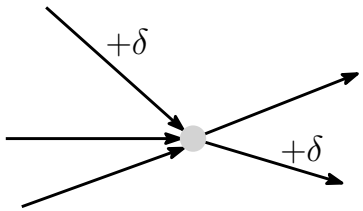
Flusserhaltungserhaltung

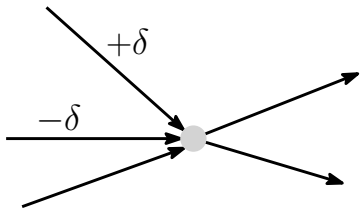
Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:

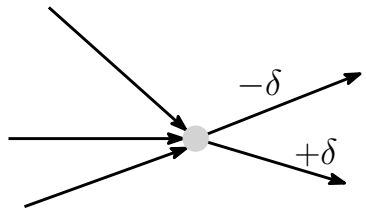


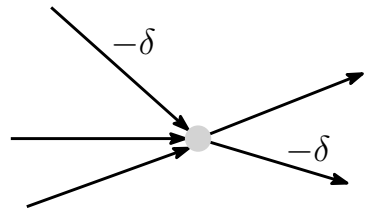
Flusserhaltungserhaltung

Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:



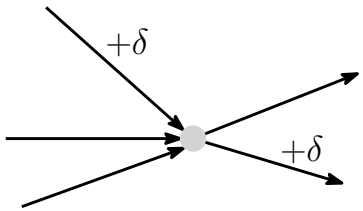


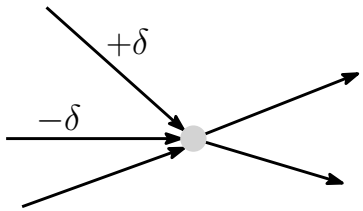


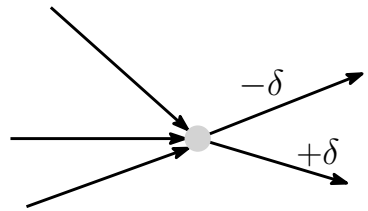


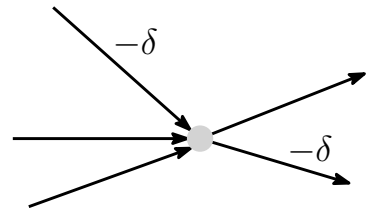
Flusserhaltungserhaltung

Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:





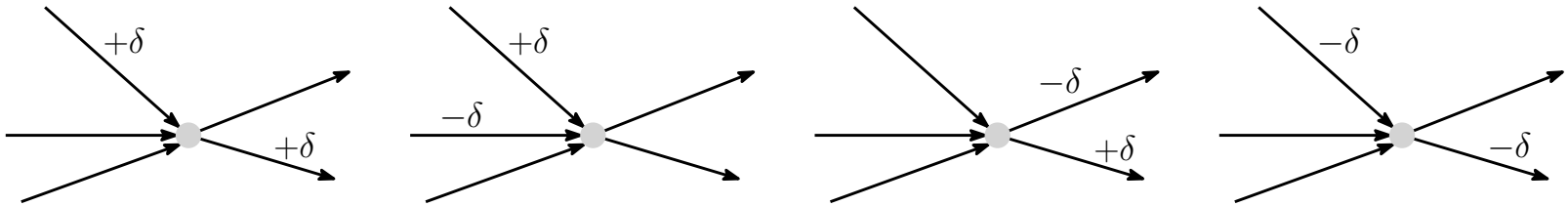




Unter Beachtung
- der Kapazität bei „$+\delta$“, und
- des aktuellen Flusses bei „$-\delta$“.

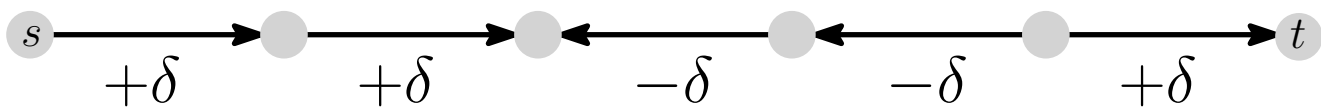
Flusserhaltungserhaltung

Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:



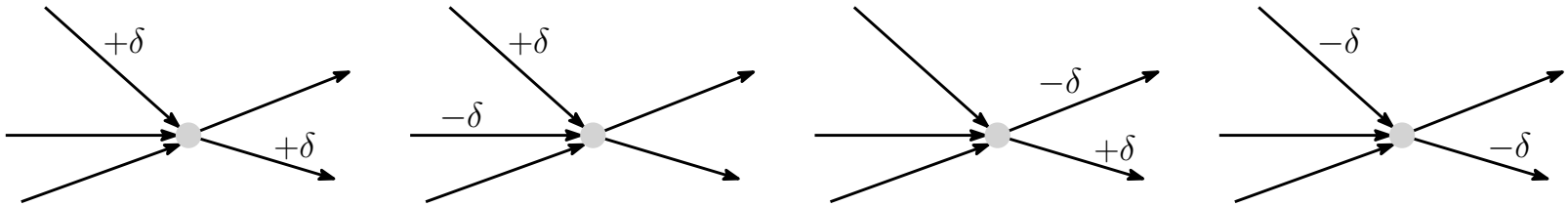
Unter Beachtung $\begin{cases} \text{der Kapazität} & \text{bei „}+\delta\text{“, und} \\ \text{des aktuellen Flusses} & \text{bei „}-\delta\text{“.} \end{cases}$

Quelle **Flussaugmentierung** Senke



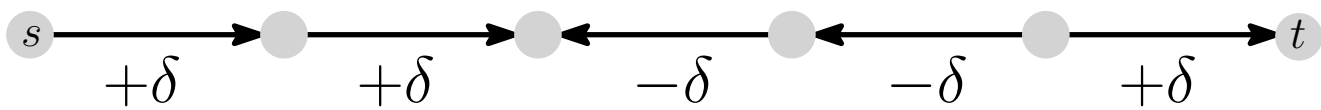
Flusserhaltungserhaltung

Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:



Unter Beachtung $\begin{cases} \text{der Kapazität} & \text{bei „}+\delta\text{“, und} \\ \text{des aktuellen Flusses} & \text{bei „}-\delta\text{“.} \end{cases}$

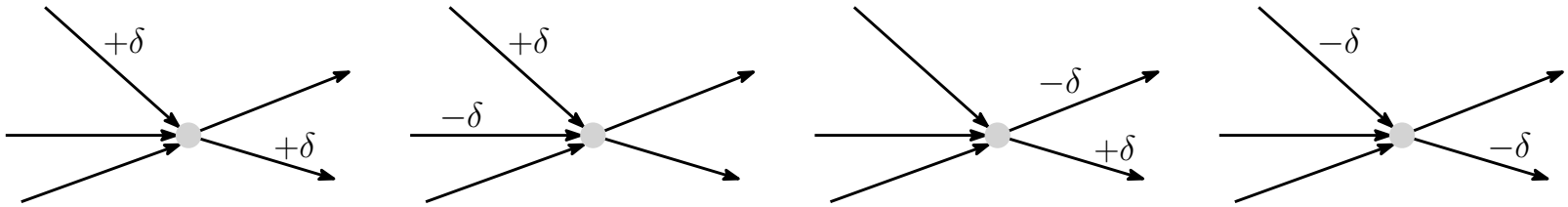
Quelle **Flussaugmentierung** Senke



augmentierender Pfad (ungerichteter Pfad!)

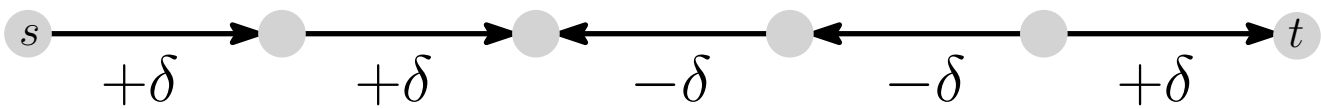
Flusserhaltungserhaltung

Lokale Veränderungen des Flusses, die die Flusserhaltung erhalten:



Unter Beachtung $\begin{cases} \text{der Kapazität} & \text{bei „}+\delta\text{“, und} \\ \text{des aktuellen Flusses} & \text{bei „}-\delta\text{“.} \end{cases}$

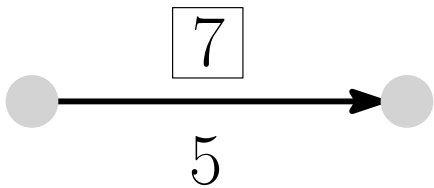
Quelle **Flussaugmentierung** Senke



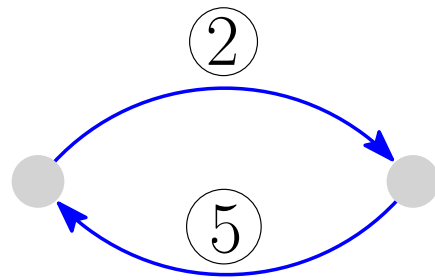
augmentierender Pfad (ungerichteter Pfad!)

Wie finden wir augmentierende Pfade?

Verwaltung des potentiellen Extraflusses/Spielraum

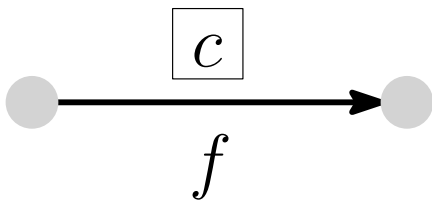


Netzwerk

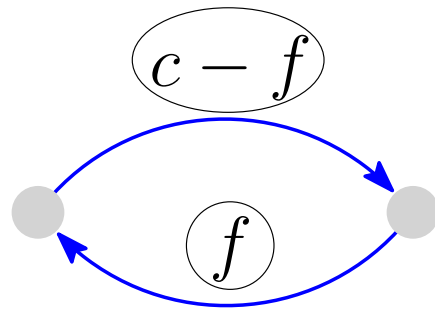


Spielraum

Verwaltung des potentiellen Extraflusses/Spielraum



Netzwerk



Spielraum

Restnetzwerk

Für $e = (u, v)$, sei $e^{\text{opp}} := (v, u)$ (entgegen gerichtete Kante).

Sei $N = (V, A, c, s, t)$ ein Netzwerk ohne entgegen gerichtete Kanten[1] und sei f ein Fluss in N. Das **Restnetzwerk** $N_f := (V, A_f, r_f, s, t)$ ist wie folgt definiert:

1. Ist $e \in A$ mit $f(e) < c(e)$, dann ist e eine Kante in A_f, mit
$$r_f(e) := c(e) - f(e).$$
2. Ist $e \in A$ mit $f(e) > 0$, dann ist e^{opp} in A_f, mit
$$r_f(e^{\text{opp}}) = f(e).$$
3. A_f enthält nur Kanten wie in (1) und (2).

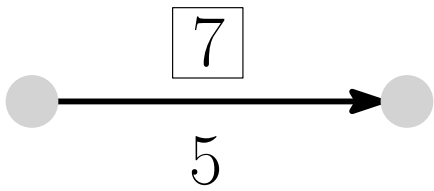
$r_f(e)$, $e \in A_f$, nennen wir die **Restkapazität** der Kante e.

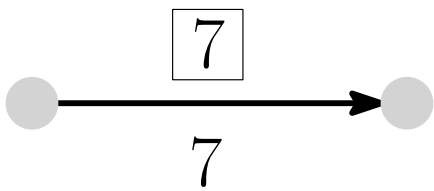
Restkapazität = „Spielraum“

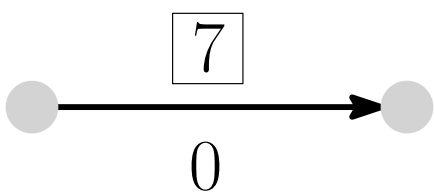
[1] Vereinfachende Annahme, ist aber nicht essentiell.

Restnetzwerk

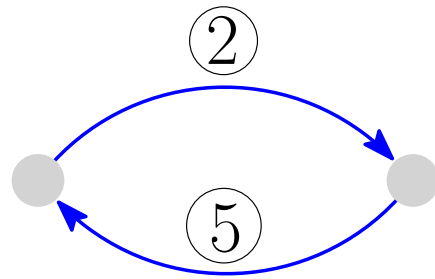
Netzwerk

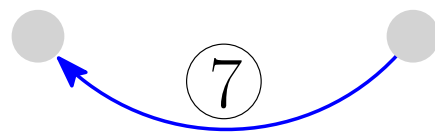


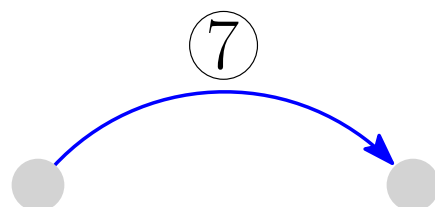




Restnetzwerk
Restkapazität







Charakterisierung maximaler Fluss

Satz

Sei N ein Netzwerk (ohne entgegegen gerichtete Kanten).

Ein Fluss f ist maximaler Fluss

$\Leftrightarrow$

es im Restnetzwerk N_f keinen gerichteten s-t-Pfad gibt.

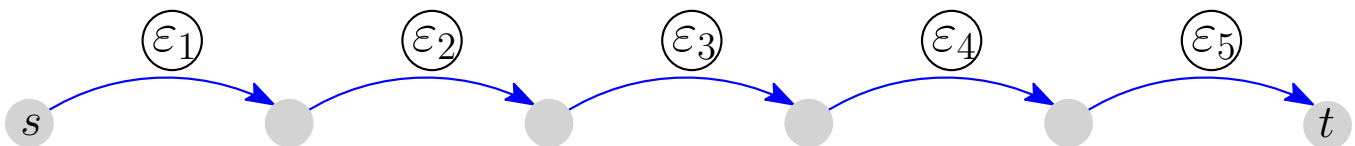
Für jeden maximalen Fluss f

gibt es einen s-t-Schnitt (S, T) mit $\mathrm{val}(f) = \mathrm{cap}(S, T)$.

Beweis

Es gibt im Restnetzwerk N_f einen gerichteten s-t-Pfad
$\Rightarrow$ f kann augmentiert werden ($\Rightarrow$ f ist nicht maximal)

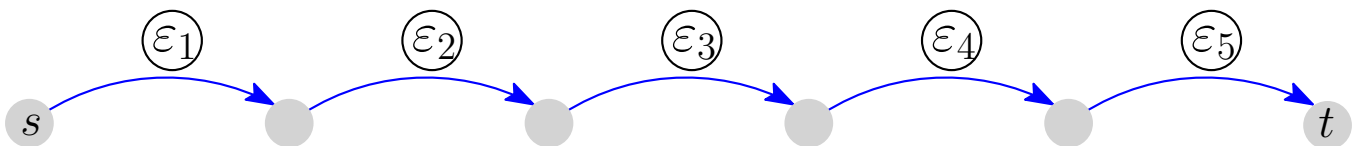
Wir betrachten einen gerichteten s-t-Pfad in N_f:



Beweis

Es gibt im Restnetzwerk N_f einen gerichteten s-t-Pfad
$\Rightarrow$ f kann augmentiert werden ($\Rightarrow$ f ist nicht maximal)

Wir betrachten einen gerichteten s-t-Pfad in N_f:

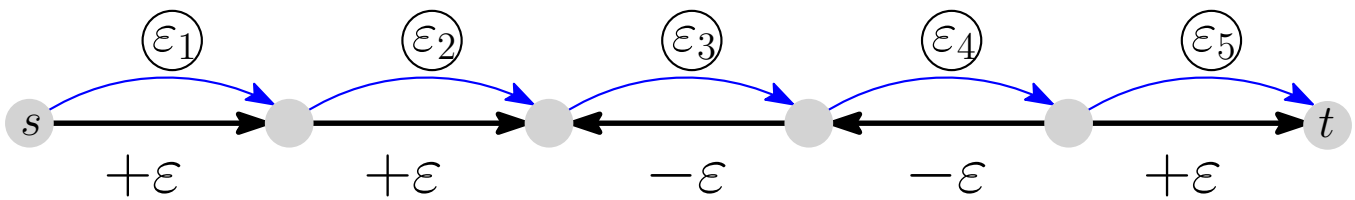


Bestimme die kleinste Restkapazität $\varepsilon := \min_i \varepsilon_i$

Beweis

Es gibt im Restnetzwerk N_f einen gerichteten s-t-Pfad
$\Rightarrow$ f kann augmentiert werden ($\Rightarrow$ f ist nicht maximal)

Wir betrachten einen gerichteten s-t-Pfad in N_f:



Bestimme die kleinste Restkapazität $\varepsilon := \min_i \varepsilon_i$

Augmentiere f entlang des Pfades um ε.

Beweis

Es gibt im Restnetzwerk N_f keinen gerichteten s-t-Pfad
$\Rightarrow$ $\exists$ s-t-Schnitt (S, T) mit $\text{cap}(S, T) = \text{val}(f)$ ($\Rightarrow$ f ist maximal)

Beweis

Es gibt im Restnetzwerk N_f keinen gerichteten s-t-Pfad
$\Rightarrow$ $\exists$ s-t-Schnitt (S, T) mit $\mathrm{cap}(S, T) = \mathrm{val}(f)$ ($\Rightarrow$ f ist maximal)

$S :=$ in N_f von s aus erreichbare Knoten; $T := V \setminus S$.

Beweis

Es gibt im Restnetzwerk N_f keinen gerichteten s-t-Pfad
$\Rightarrow$ $\exists$ s-t-Schnitt (S, T) mit $\text{cap}(S, T) = \text{val}(f)$ ($\Rightarrow$ f ist maximal)

$S :=$ in N_f von s aus erreichbare Knoten; $T := V \setminus S$.

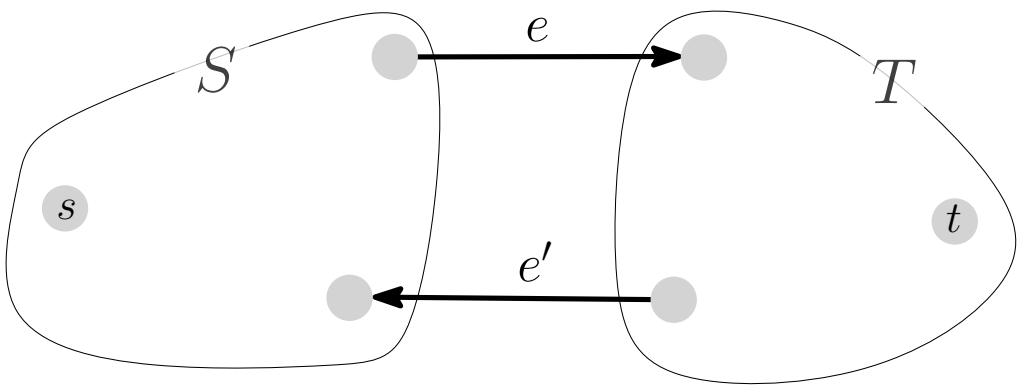
$$\left.\begin{array}{l} s \text{ von } s \text{ aus in } N_f \text{ erreichbar} \Rightarrow s \in S \\ t \text{ von } s \text{ aus nicht erreichbar} \Rightarrow t \notin S \end{array}\right\} \Rightarrow (S, T) \text{ ist } s\text{-}t\text{-Schnitt.}$$

Beweis

Es gibt im Restnetzwerk N_f keinen gerichteten s-t-Pfad
$\Rightarrow$ $\exists$ s-t-Schnitt (S, T) mit $\text{cap}(S, T) = \text{val}(f)$ ($\Rightarrow$ f ist maximal)

$S :=$ in N_f von s aus erreichbare Knoten; $T := V \setminus S$.

$$\left.\begin{array}{l} s \text{ von } s \text{ aus in } N_f \text{ erreichbar} \Rightarrow s \in S \\ t \text{ von } s \text{ aus nicht erreichbar} \Rightarrow t \notin S \end{array}\right\} \Rightarrow (S, T) \text{ ist } s\text{-}t\text{-Schnitt.}$$

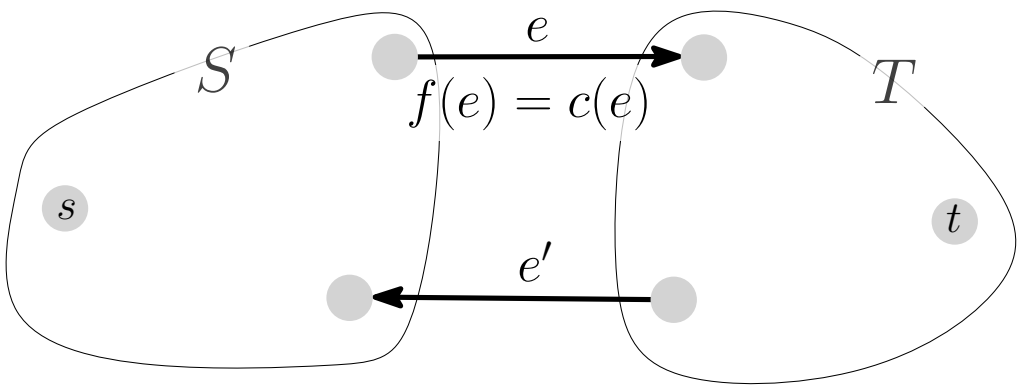


Beweis

Es gibt im Restnetzwerk N_f keinen gerichteten s-t-Pfad
$\Rightarrow$ $\exists$ s-t-Schnitt (S, T) mit $\text{cap}(S, T) = \text{val}(f)$ ($\Rightarrow$ f ist maximal)

$S :=$ in N_f von s aus erreichbare Knoten; $T := V \setminus S$.

$$\left.\begin{array}{l} s \text{ von } s \text{ aus in } N_f \text{ erreichbar} \Rightarrow s \in S \\ t \text{ von } s \text{ aus nicht erreichbar} \Rightarrow t \notin S \end{array}\right\} \Rightarrow (S, T) \text{ ist } s\text{-}t\text{-Schnitt.}$$

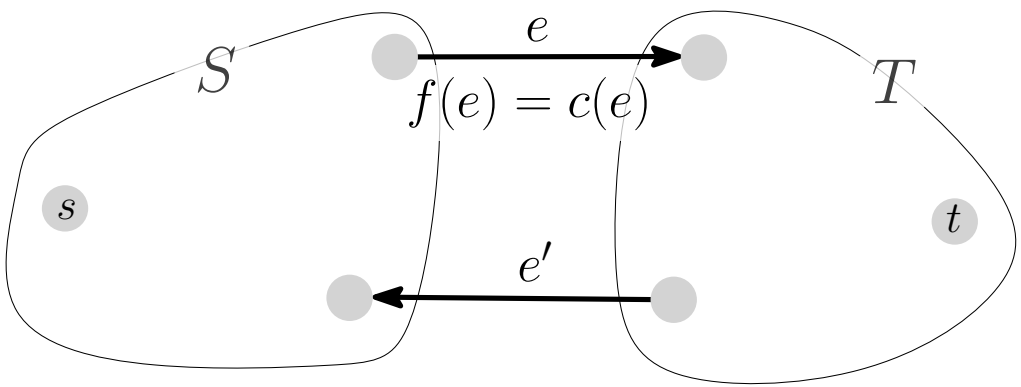


Beweis

Es gibt im Restnetzwerk N_f keinen gerichteten s-t-Pfad
$\Rightarrow$ $\exists$ s-t-Schnitt (S, T) mit $\text{cap}(S, T) = \text{val}(f)$ ($\Rightarrow$ f ist maximal)

$S :=$ in N_f von s aus erreichbare Knoten; $T := V \setminus S$.

$$\left.\begin{array}{l} s \text{ von } s \text{ aus in } N_f \text{ erreichbar} \Rightarrow s \in S \\ t \text{ von } s \text{ aus nicht erreichbar} \Rightarrow t \notin S \end{array}\right\} \Rightarrow (S, T) \text{ ist } s\text{-}t\text{-Schnitt.}$$



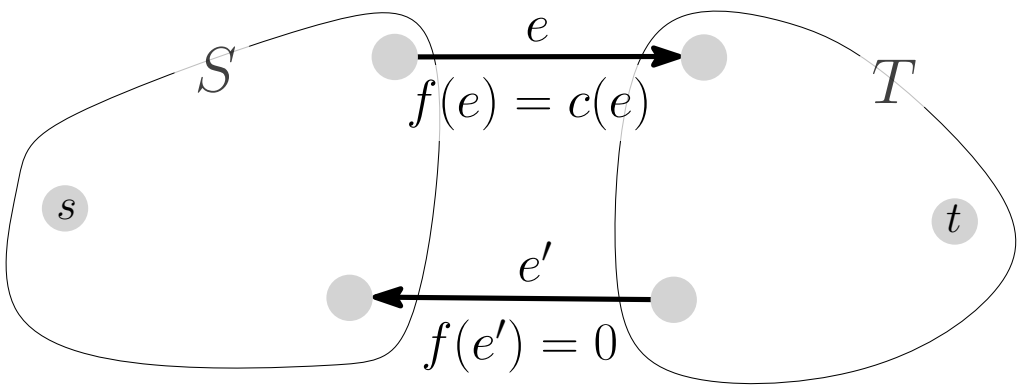
$$f(S, T) = \text{cap}(S, T)$$

Beweis

Es gibt im Restnetzwerk N_f keinen gerichteten s-t-Pfad
$\Rightarrow$ $\exists$ s-t-Schnitt (S, T) mit $\text{cap}(S, T) = \text{val}(f)$ ($\Rightarrow$ f ist maximal)

$S :=$ in N_f von s aus erreichbare Knoten; $T := V \setminus S$.

$$\left.\begin{array}{l} s \text{ von } s \text{ aus in } N_f \text{ erreichbar} \Rightarrow s \in S \\ t \text{ von } s \text{ aus nicht erreichbar} \Rightarrow t \notin S \end{array}\right\} \Rightarrow (S, T) \text{ ist } s\text{-}t\text{-Schnitt.}$$



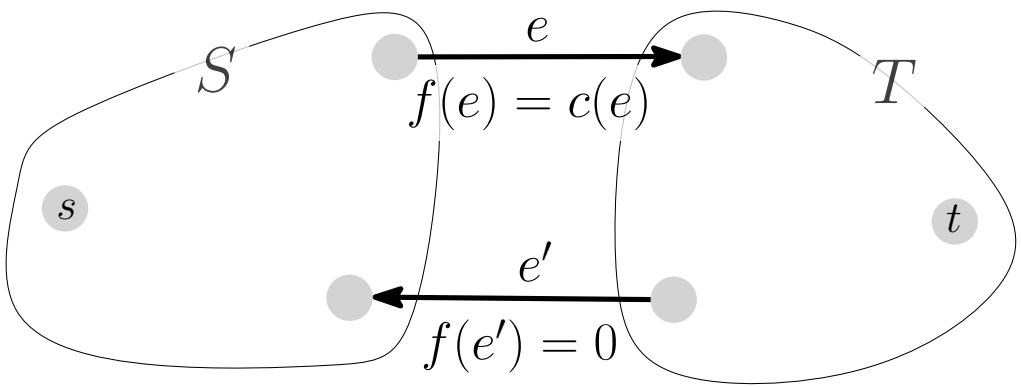
$$f(S, T) = \text{cap}(S, T)$$

Beweis

Es gibt im Restnetzwerk N_f keinen gerichteten s-t-Pfad
$\Rightarrow$ $\exists$ s-t-Schnitt (S, T) mit $\text{cap}(S, T) = \text{val}(f)$ ($\Rightarrow$ f ist maximal)

$S :=$ in N_f von s aus erreichbare Knoten; $T := V \setminus S$.

$$\left.\begin{array}{l} s \text{ von } s \text{ aus in } N_f \text{ erreichbar} \Rightarrow s \in S \\ t \text{ von } s \text{ aus nicht erreichbar} \Rightarrow t \notin S \end{array}\right\} \Rightarrow (S, T) \text{ ist } s\text{-}t\text{-Schnitt.}$$



$$f(S, T) = \text{cap}(S, T)$$

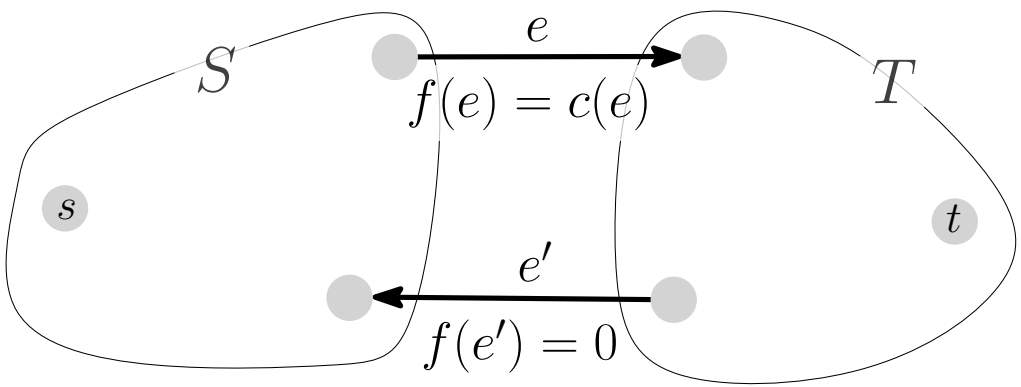
$$f(T, S) = 0$$

Beweis

Es gibt im Restnetzwerk N_f keinen gerichteten s-t-Pfad
$\Rightarrow$ $\exists$ s-t-Schnitt (S, T) mit $\text{cap}(S, T) = \text{val}(f)$ ($\Rightarrow$ f ist maximal)

$S :=$ in N_f von s aus erreichbare Knoten; $T := V \setminus S$.

$$\left.\begin{array}{l} s \text{ von } s \text{ aus in } N_f \text{ erreichbar} \Rightarrow s \in S \\ t \text{ von } s \text{ aus nicht erreichbar} \Rightarrow t \notin S \end{array}\right\} \Rightarrow (S, T) \text{ ist } s\text{-}t\text{-Schnitt.}$$

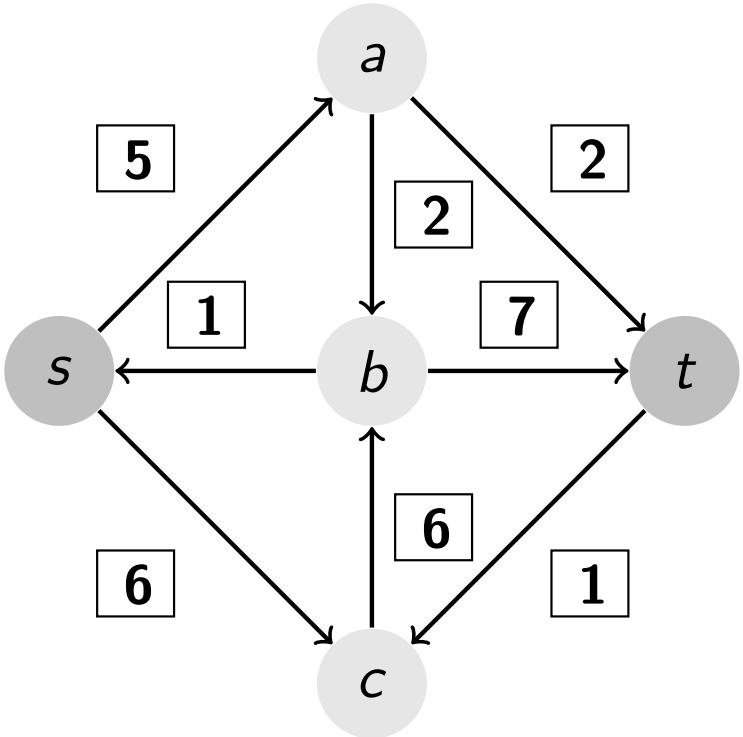


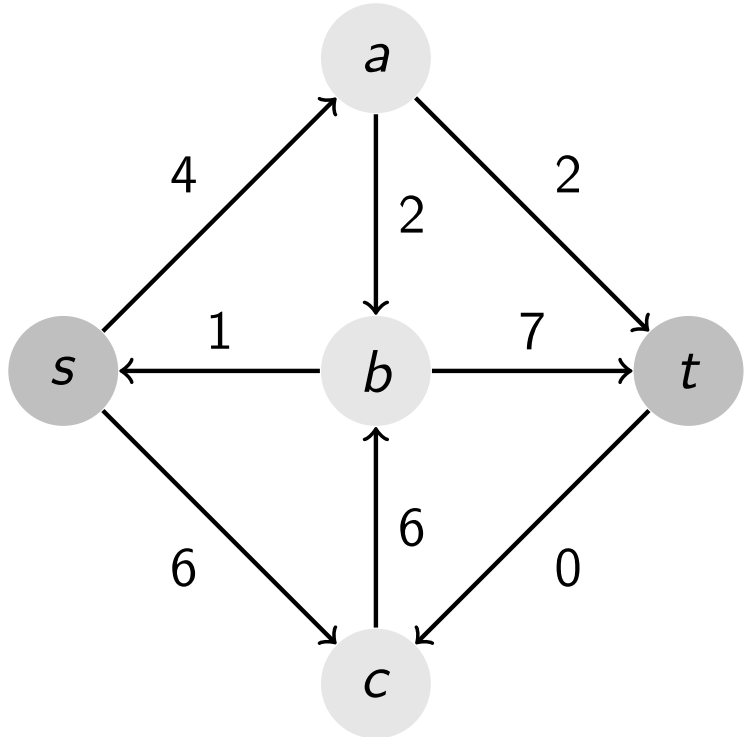
$$f(S, T) = \text{cap}(S, T)$$

$$f(T, S) = 0$$

$$\text{val}(f) = \underbrace{f(S, T)}_{=\text{cap}(S,T)} - \underbrace{f(T, S)}_{=0} = \text{cap}(S, T)$$

Beweis – Beispiel „Finde den Schnitt“





Fluss mit Wert $4 - 1 + 6 = 9$

Charakterisierung maximaler Fluss

Satz

Sei N ein Netzwerk (ohne entgegegen gerichtete Kanten).

Ein Fluss f ist maximaler Fluss

$$\Leftrightarrow$$

es im Restnetzwerk N_f keinen gerichteten s-t-Pfad gibt.

Für jeden maximalen Fluss f

gibt es einen s-t-Schnitt (S, T) mit $\mathrm{val}(f) = \mathrm{cap}(S, T)$.

Charakterisierung maximaler Fluss

Satz

Sei N ein Netzwerk (ohne entgegegen gerichtete Kanten).

Ein Fluss f ist maximaler Fluss

$\Leftrightarrow$

es im Restnetzwerk N_f keinen gerichteten s-t-Pfad gibt.

Für jeden maximalen Fluss f

gibt es einen s-t-Schnitt (S, T) mit $\mathrm{val}(f) = \mathrm{cap}(S, T)$.

- Zeigt noch nicht, dass es immer einen maximalen Fluss gibt.

Ford-Fulkerson Algorithmus

Ford-Fulkerson(V, A, c, s, t)

1: $f \leftarrow \mathbf{0}$ ▷ Fluss konstant 0
2: **while** $\exists$ s-t-Pfad P in N_f **do** ▷ augmentierender Pfad
3: Augmentiere den Fluss entlang P
4: **return** f ▷ maximaler Fluss

Ford-Fulkerson Algorithmus

Ford-Fulkerson(V, A, c, s, t)

1: $f \leftarrow \mathbf{0}$ ▷ Fluss konstant 0
2: **while** $\exists$ s-t-Pfad P in N_f **do** ▷ augmentierender Pfad
3: Augmentiere den Fluss entlang P
4: **return** f ▷ maximaler Fluss

- Wir können nicht garantieren, dass der Algorithmus terminiert.

Ford-Fulkerson Algorithmus

Ford-Fulkerson(V, A, c, s, t)

1: $f \leftarrow \mathbf{0}$ ▷ Fluss konstant 0
2: **while** $\exists$ s-t-Pfad P in N_f **do** ▷ augmentierender Pfad
3: Augmentiere den Fluss entlang P
4: **return** f ▷ maximaler Fluss

- Wir können nicht garantieren, dass der Algorithmus terminiert.
- Der Algorithmus kann bei Kapazitäten aus $\mathbb{R}$ unendlich laufen.

Ford-Fulkerson Algorithmus

Ford-Fulkerson(V, A, c, s, t)

1: $f \leftarrow \mathbf{0}$ ▷ Fluss konstant 0
2: **while** $\exists$ s-t-Pfad P in N_f **do** ▷ augmentierender Pfad
3: Augmentiere den Fluss entlang P
4: **return** f ▷ maximaler Fluss

- Wir können nicht garantieren, dass der Algorithmus terminiert.
- Der Algorithmus kann bei Kapazitäten aus $\mathbb{R}$ unendlich laufen.
- Bei Kapazitäten aus $\mathbb{N}_0$ bleiben im Algorithmus Flüsse und Restkapazitäten ganzzahlig. In jedem Augmentierungsschritt wird der Fluss ganzzahlig ≥ 1 verbessert. D.h. insbesondere auch, dass das Ergebnis **ganzzahlig** ($A \to \mathbb{N}_0$) ist.

Analyse

Sei $n := |V|$ und $m := |A|$ für Netzwerk $N = (V, A, c, s, t)$.

[2] Vereinfachende Annahme, ist aber nicht essentiell.

Analyse

Sei $n := |V|$ und $m := |A|$ für Netzwerk $N = (V, A, c, s, t)$.

- Angenommen $c\colon A \to \mathbb{N}_0$ und $U := \max_{e \in A} c(e)$. Dann gilt
$$\mathrm{val}(f) \leq \mathrm{cap}(\{s\}, V \setminus \{s\}) \leq (n-1)U$$
und es gibt **höchstens** $(n-1)U$ **Augmentierungsschritte**.

[2] Vereinfachende Annahme, ist aber nicht essentiell.

Analyse

Sei $n := |V|$ und $m := |A|$ für Netzwerk $N = (V, A, c, s, t)$.

- Angenommen $c\colon A \to \mathbb{N}_0$ und $U := \max_{e \in A} c(e)$. Dann gilt
$$\mathrm{val}(f) \leq \mathrm{cap}(\{s\}, V \setminus \{s\}) \leq (n-1)U$$
und es gibt **höchstens** $(n-1)U$ **Augmentierungsschritte**.
- **Ein Augmentierungsschritt**
Suche s-t-Pfad in N_f, Augmentieren, Aktualisierung von N_f **benötigt** $O(m)$ **Zeit**.

[2] Vereinfachende Annahme, ist aber nicht essentiell.

Analyse

Sei $n := |V|$ und $m := |A|$ für Netzwerk $N = (V, A, c, s, t)$.

- Angenommen $c\colon A \to \mathbb{N}_0$ und $U := \max_{e \in A} c(e)$. Dann gilt
$$\mathrm{val}(f) \leq \mathrm{cap}(\{s\}, V \setminus \{s\}) \leq (n-1)U$$
und es gibt **höchstens** $(n-1)U$ **Augmentierungsschritte**.
- **Ein Augmentierungsschritt**
Suche s-t-Pfad in N_f, Augmentieren, Aktualisierung von N_f **benötigt** $O(m)$ **Zeit**.

Satz (Ford-Fulkerson mit ganzzahligen Kapazitäten)

Sei $N = (V, A, c, s, t)$ ein Netzwerk mit $c : A \to \mathbb{N}_0^{\leq U}$, $U \in \mathbb{N}$, ohne entgegen gerichtete Kanten.[2] *Dann gibt es einen ganzzahligen maximalen Fluss. Er kann in Zeit $O(mnU)$ berechnet werden.*

[2] Vereinfachende Annahme, ist aber nicht essentiell.

Maxflow-Mincut Theorem

Damit haben wir auch bewiesen.

Satz („Maxflow-Mincut Theorem“, ganzzahlig)

Jedes Netzwerk ohne entgegen gerichtete Kanten mit ganzzahligen Kapazitäten erfüllt

$$\max_{f \text{ Fluss}} \mathrm{val}(f) = \min_{(S,T) \text{ s-t-Schnitt}} \mathrm{cap}(S,T) \ .$$

Maxflow-Mincut Theorem

Damit haben wir auch bewiesen.

Satz („Maxflow-Mincut Theorem“, ganzzahlig)

Jedes Netzwerk ohne entgegen gerichtete Kanten mit ganzzahligen Kapazitäten erfüllt

$$\max_{f \text{ Fluss}} \mathrm{val}(f) = \min_{(S,T) \text{ s-t-Schnitt}} \mathrm{cap}(S,T) \ .$$

Der Satz gilt auch, wenn das Netzwerk entgegen gerichtete Kanten hat.

Maxflow-Mincut Theorem

Damit haben wir auch bewiesen.

Satz („Maxflow-Mincut Theorem“, ganzzahlig)

Jedes Netzwerk ohne entgegen gerichtete Kanten mit ganzzahligen Kapazitäten erfüllt

$$\max_{f \text{ Fluss}} \mathrm{val}(f) = \min_{(S,T) \text{ s-t-Schnitt}} \mathrm{cap}(S,T) \ .$$

Der Satz gilt auch, wenn das Netzwerk entgegen gerichtete Kanten hat. Und er gilt auch bei beliebigen reellen Kapazitäten.

Anmerkungen

- **Capacity-Scaling** [Dinitz-Gabow'73] Sind in einem Netzwerk alle Kapazitäten ganzzahlig und höchstens U, so kann ein ganzzahliger maximaler Fluss in Zeit $O(mn(1 + \log U))$ berechnet werden kann.

Anmerkungen

- **Capacity-Scaling** [Dinitz-Gabow'73] Sind in einem Netzwerk alle Kapazitäten ganzzahlig und höchstens U, so kann ein ganzzahliger maximaler Fluss in Zeit $O(mn(1 + \log U))$ berechnet werden kann.
- **Dynamic Trees** [Sleator-Tarjan'83] Der maximale Fluss eines Netzwerks kann in Zeit $O(mn \log n)$ berechnet werden.

Anmerkungen

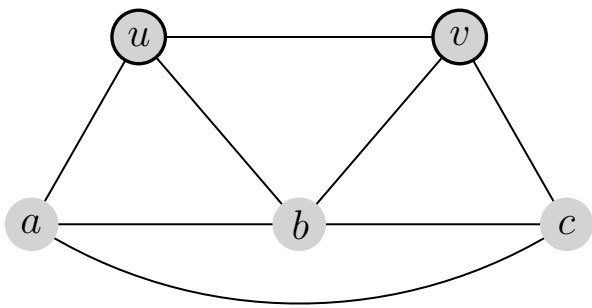
- **Capacity-Scaling** [Dinitz-Gabow'73] Sind in einem Netzwerk alle Kapazitäten ganzzahlig und höchstens U, so kann ein ganzzahliger maximaler Fluss in Zeit $O(mn(1 + \log U))$ berechnet werden kann.
- **Dynamic Trees** [Sleator-Tarjan'83] Der maximale Fluss eines Netzwerks kann in Zeit $O(mn \log n)$ berechnet werden.
- Alle Schranken gelten nach Maxflow-Mincut auch für die Berechnung eines minimalen s-t-Schnitts.

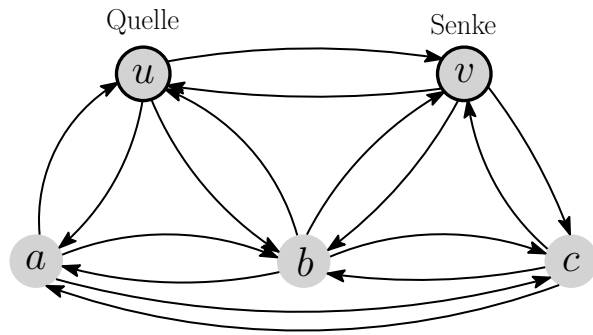
Anmerkungen

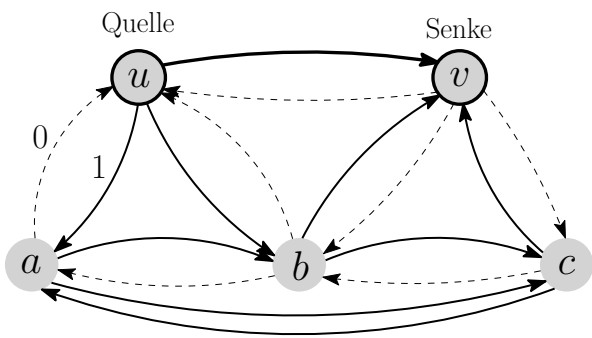
- **Capacity-Scaling** [Dinitz-Gabow'73] Sind in einem Netzwerk alle Kapazitäten ganzzahlig und höchstens U, so kann ein ganzzahliger maximaler Fluss in Zeit $O(mn(1 + \log U))$ berechnet werden.
- **Dynamic Trees** [Sleator-Tarjan'83] Der maximale Fluss eines Netzwerks kann in Zeit $O(mn \log n)$ berechnet werden.
- Alle Schranken gelten nach Maxflow-Mincut auch für die Berechnung eines minimalen s-t-Schnitts.

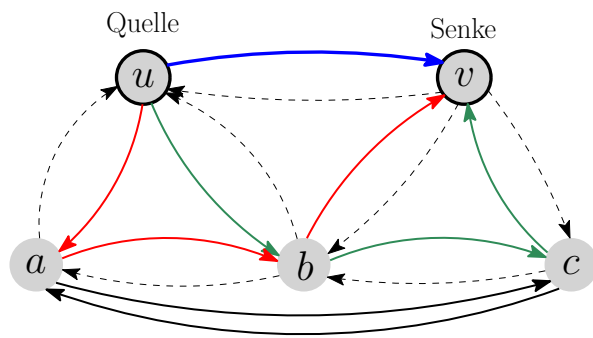
- Wir besprechen als Nächstes weitere Anwendungen (Matchings, Bildsegmentierung).

Flüsse in Netzwerken: Anwendungen (Teil 1)









I Matchings – Problemstellung

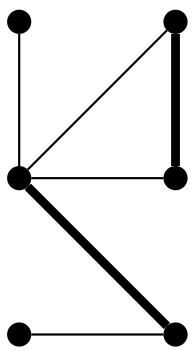
Maximum Bipartite Matching Problem. Gegeben ein bipartiter Graph, finde ein maximum (d.h. kardinalitätsmaximales) Matching.

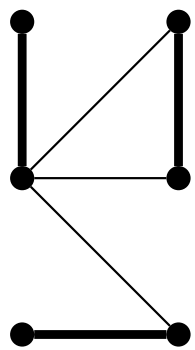
Graph, ungerichtet, ungewichtet.

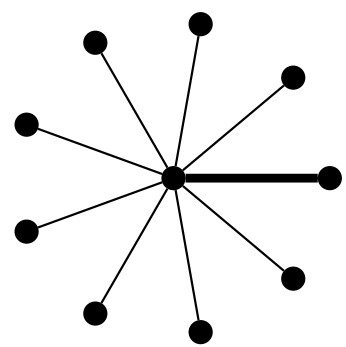
Matching – Definition

Eine Kantenmenge $M \subseteq E$ heisst **Matching** in einem Graphen $G = (V, E)$, falls kein Knoten des Graphen zu mehr als einer Kante aus M inzident ist, d.h., wenn

$$e \cap f = \emptyset \quad \text{für alle } e,f \in M \text{ mit } e \neq f.$$



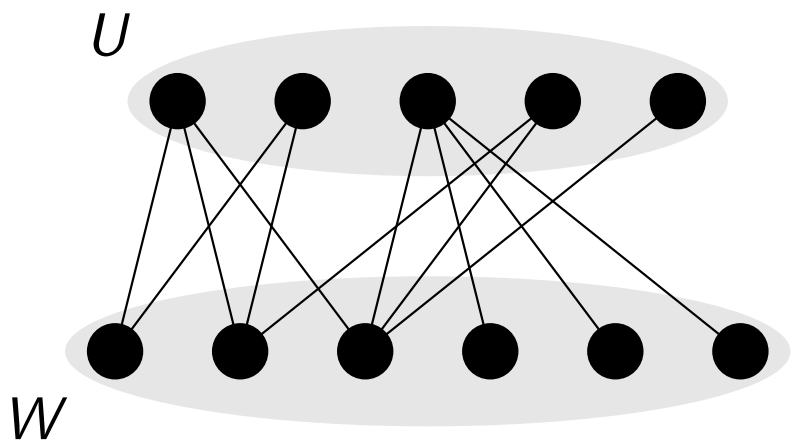




Bipartiter Graph – Definition

Die Knotenmenge eines **bipartiten Graphen** $G = (U \uplus W, E)$ besteht aus zwei disjunkten Mengen U und W und die Kanten von G verlaufen nur zwischen den beiden Mengen, d.h.

$$\forall e \in E\colon\ |e \cap U| = |e \cap W| = 1\ .$$



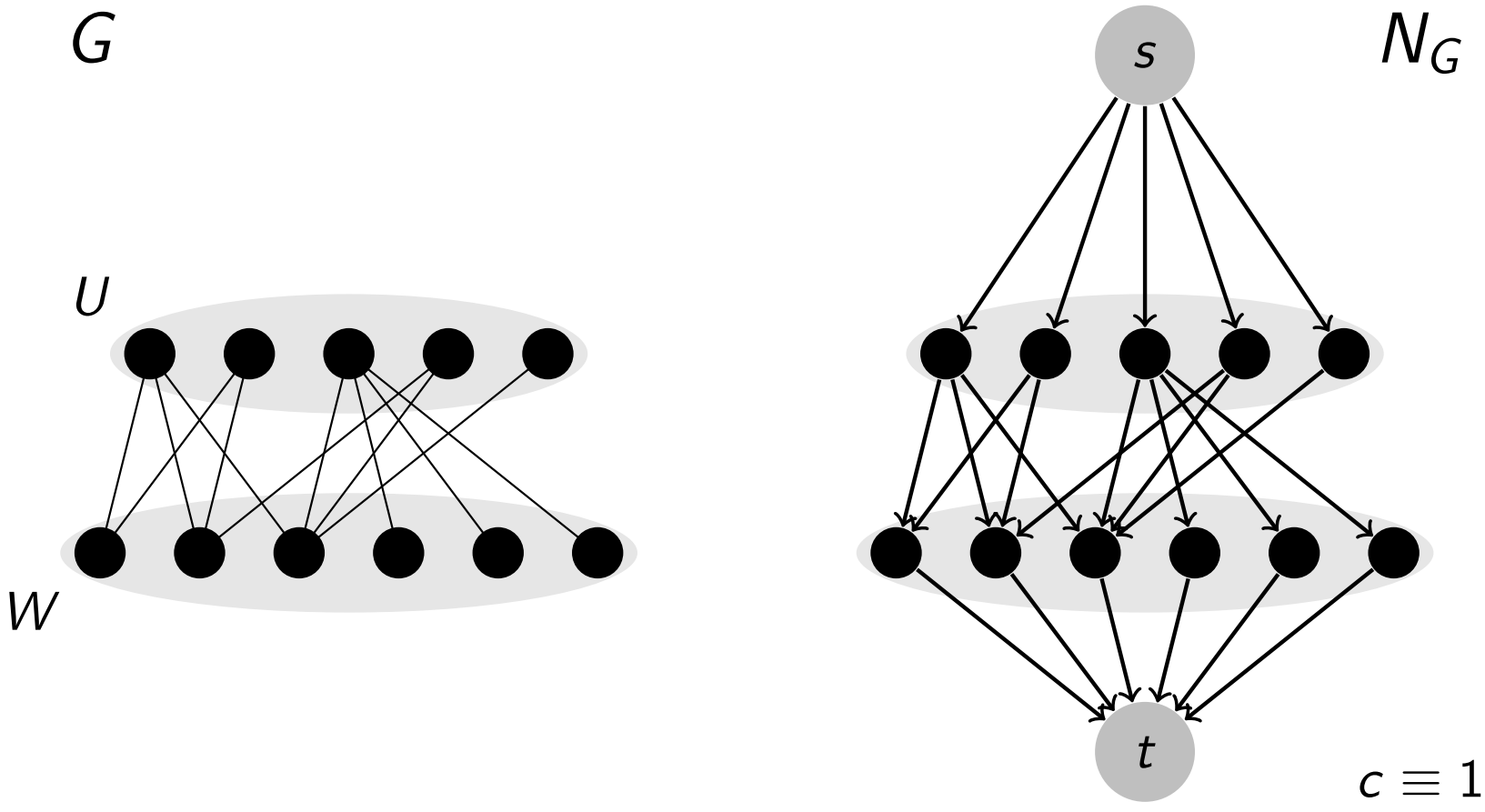
Graph zu Netzwerk (für Matchings)

Wir bilden jeden bipartiten Graphen (mit vorgegebener Knotenpartition $U \uplus W$) auf ein Netzwerk ab.

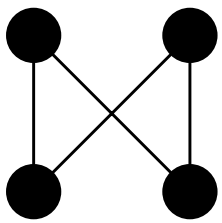
$$\overbrace{G = (U \uplus W, E)}^{\text{bipartiter Graph}} \quad \mapsto \quad \overbrace{N_G = (\underbrace{U \uplus W \uplus \{s,t\}}_{\text{Knotenmenge}}, A, c, s, t)}^{\text{Netzwerk}}$$

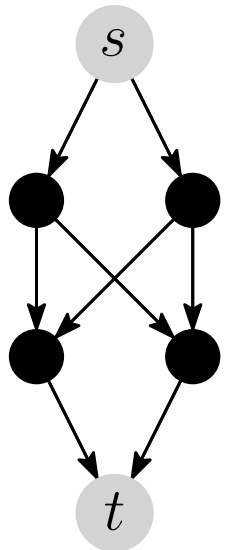
- $s \neq t$ zusätzliche Knoten.
- $A := \{s\} \times U \ \cup \ \{(u,w) \in U \times W \mid \{u,w\} \in E\} \ \cup \ W \times \{t\}$.
- $c \equiv 1$.

Graph zu Netzwerk (für Matchings)



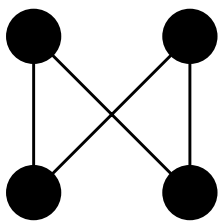
Maximale Flüsse in N_G

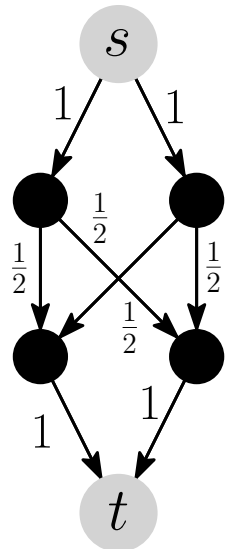




$c \equiv 1$

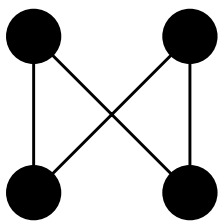
Maximale Flüsse in N_G

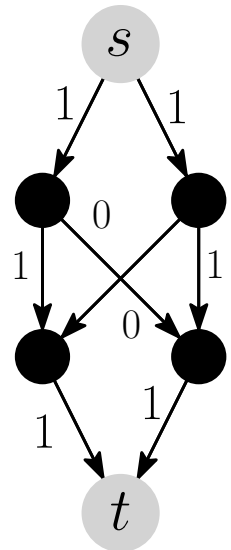




$c \equiv 1$

Maximale Flüsse in N_G





$c \equiv 1$

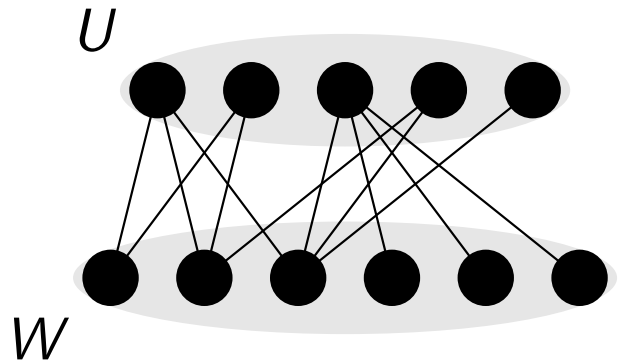
$\Leftarrow$

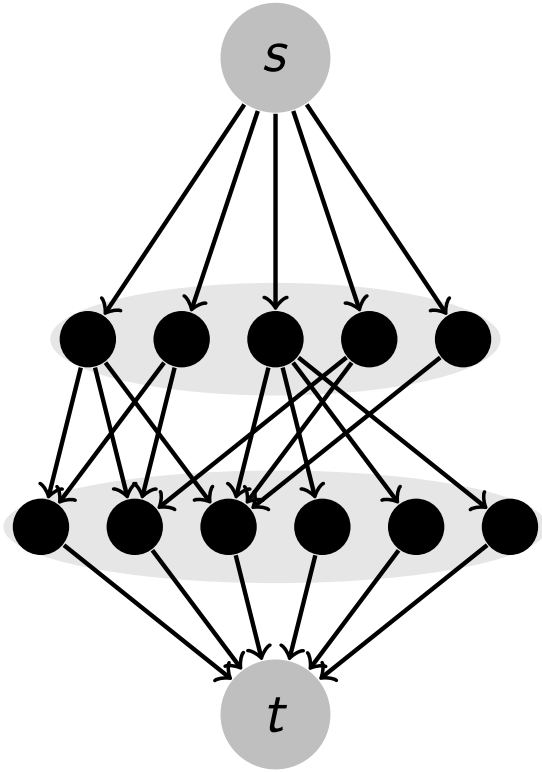
Satz (Ford-Fulkerson, ganzzahlig)

Sei N ein Netzwerk mit ganzz. Kapazitäten $\leq U$. Dann gibt es einen **ganzz. maximalen Fluss**. *Er kann in* **Zeit** $O(mnU)$ *berechnet werden.*

Hier: $U = 1$.

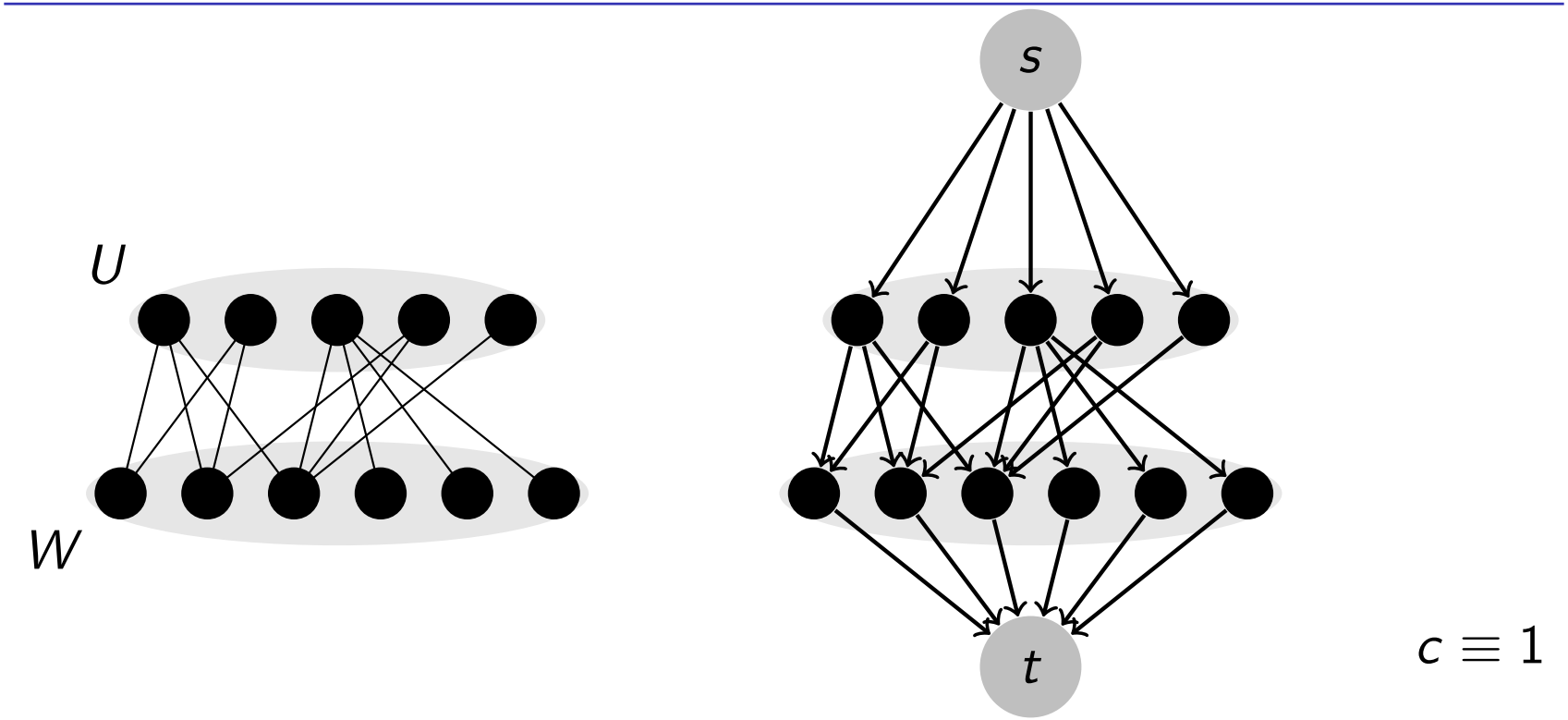
Matching zu Fluss – und vice versa





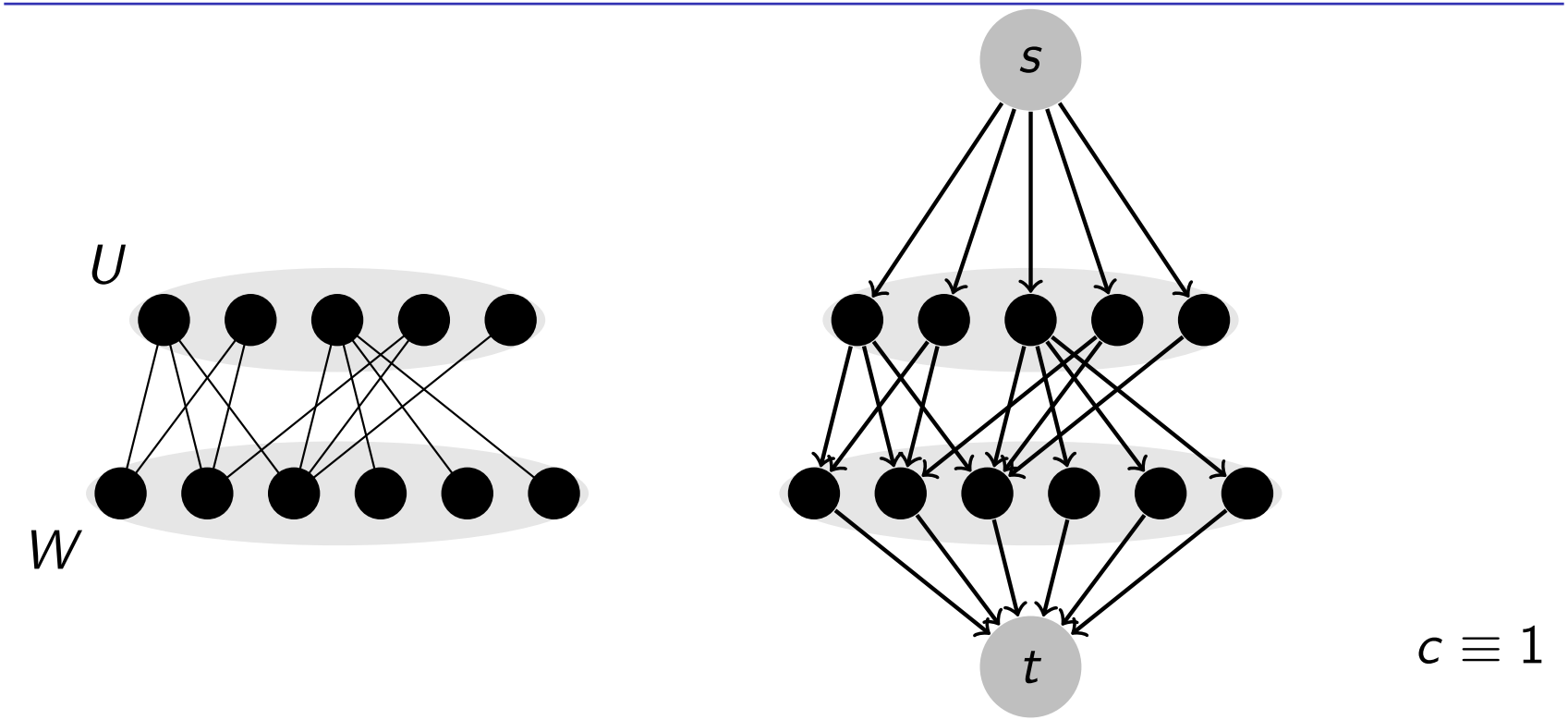
$c \equiv 1$

Matching zu Fluss – und vice versa



Matching M in G $\quad\mapsto\quad$ Fluss f_M in N_G mit $\mathrm{val}(f_M) = |M|$.

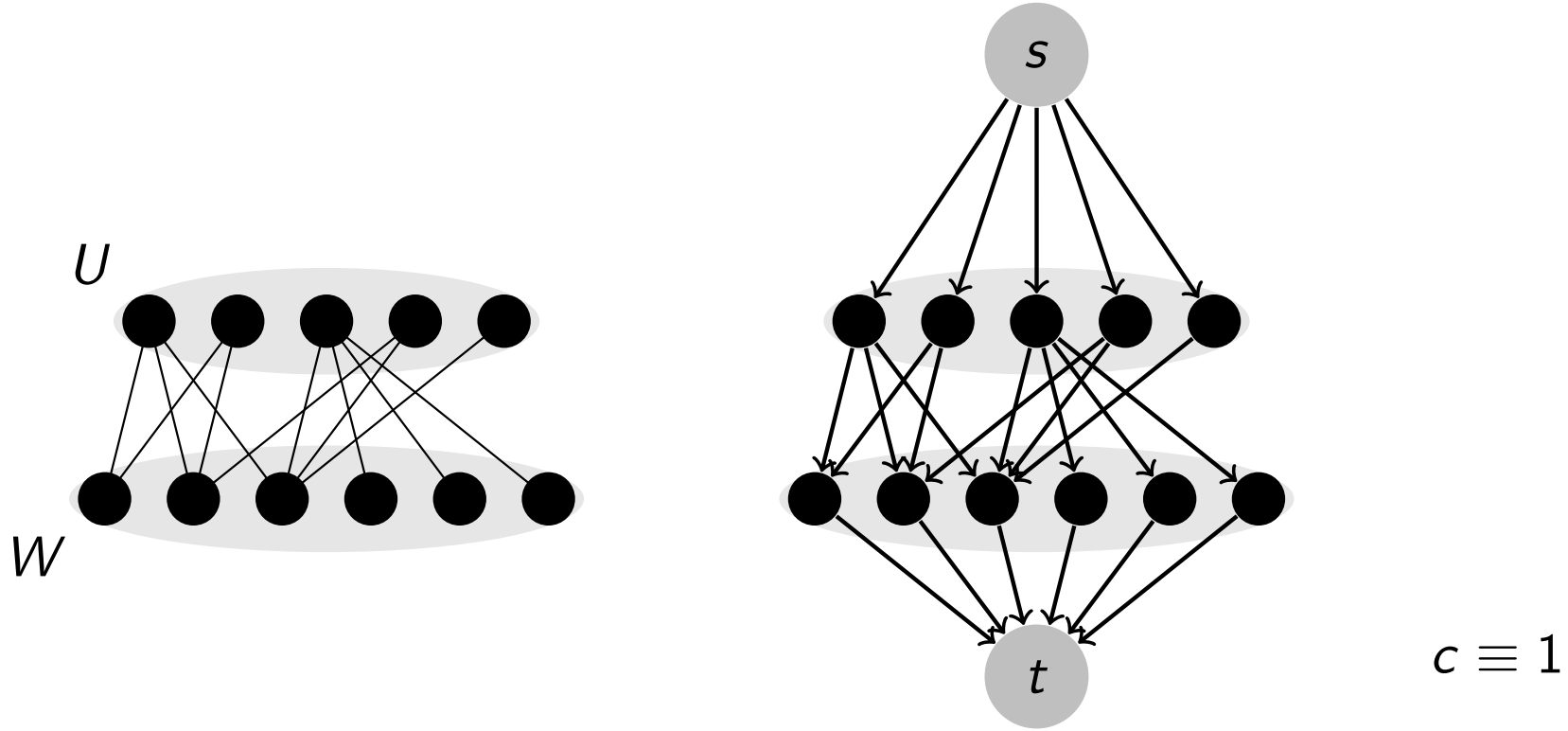
Matching zu Fluss – und vice versa



Matching M in G $\quad\mapsto\quad$ Fluss f_M in N_G mit $\mathrm{val}(f_M) = |M|$.

Ganzz. Fluss f in N_G $\quad\mapsto\quad$ Matching M in G mit $|M| = \mathrm{val}(f)$.

Matching zu Fluss – und vice versa



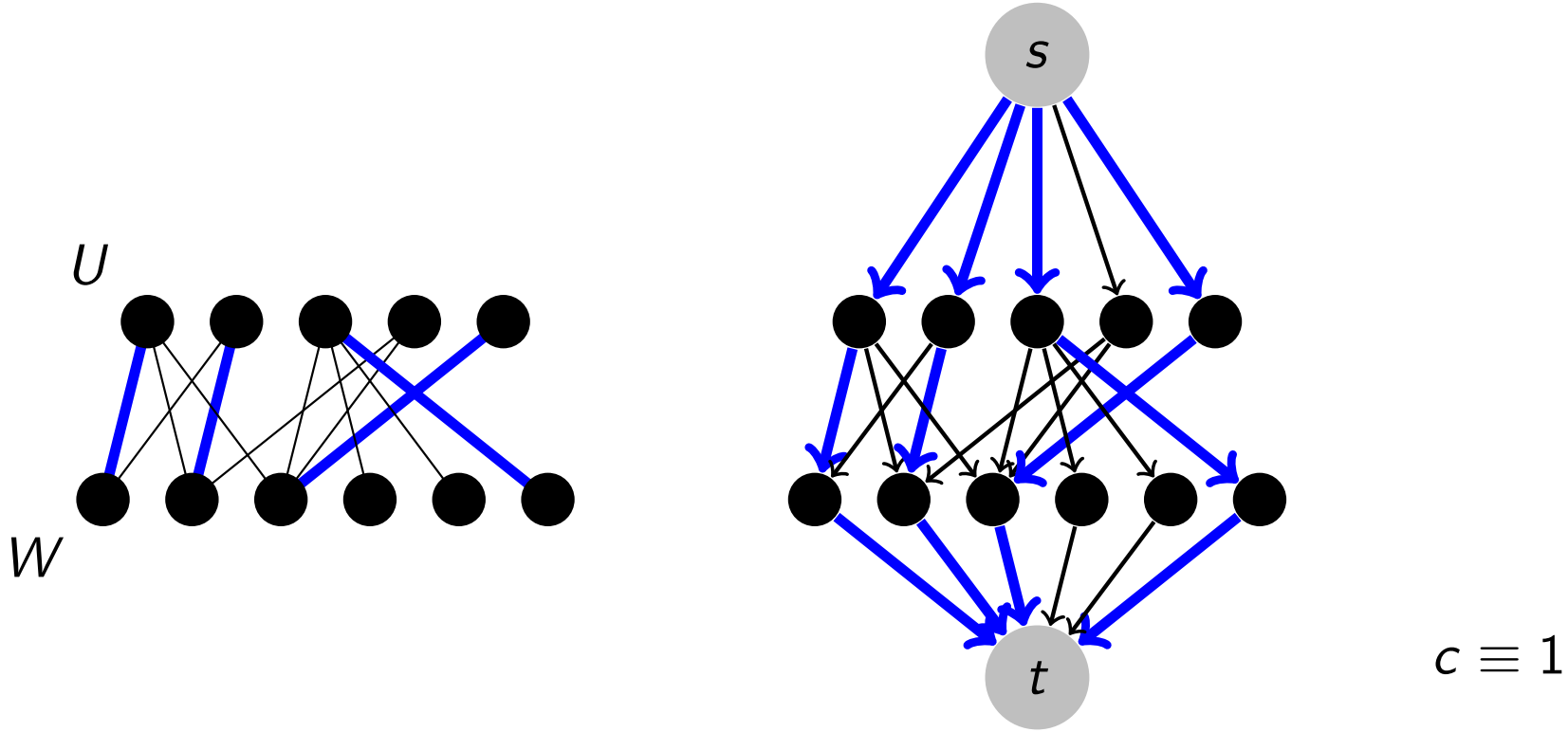
Matching M in $G \quad \mapsto \quad$ Fluss f_M in N_G mit $\mathrm{val}(f_M) = |M|$.

Ganzz. Fluss f in $N_G \quad \mapsto \quad$ Matching M in G mit $|M| = \mathrm{val}(f)$.

Maximum Matching in G „$\simeq$“ ganzz. Maxflow in N_G.

$$\max_{M \text{ Matching in } G} |M| = \max_{f \text{ Fluss in } N_G} \mathrm{val}(f)$$

Matching zu Fluss – und vice versa



Matching M in $G \quad \mapsto \quad$ Fluss f_M in N_G mit $\mathrm{val}(f_M) = |M|$.

Ganzz. Fluss f in $N_G \quad \mapsto \quad$ Matching M in G mit $|M| = \mathrm{val}(f)$.

Maximum Matching in G „$\simeq$“ ganzz. Maxflow in N_G.

$$\max_{M \text{ Matching in } G} |M| = \max_{f \text{ Fluss in } N_G} \mathrm{val}(f)$$

II Kantendisjunkte Pfade

Kantendisjunkte Pfade Problem. Gegeben ein Graph G mit zwei ausgezeichneten Knoten u und v, $v \neq u$, bestimme eine möglichst grosse Menge kantendisjunkter u-v-Pfade.

Zur Erinnerung:

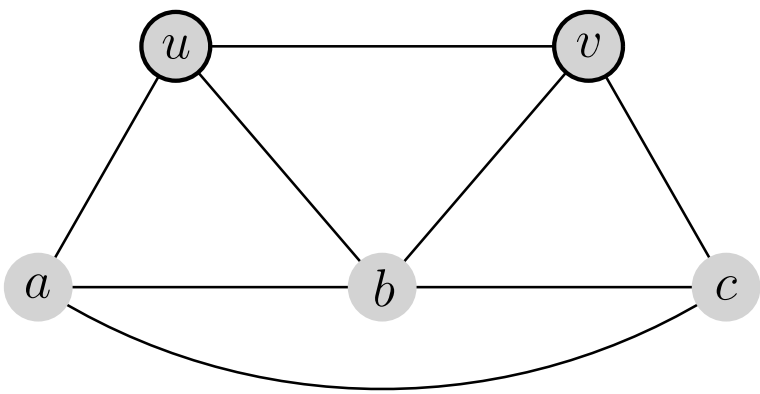
Satz (Menger)

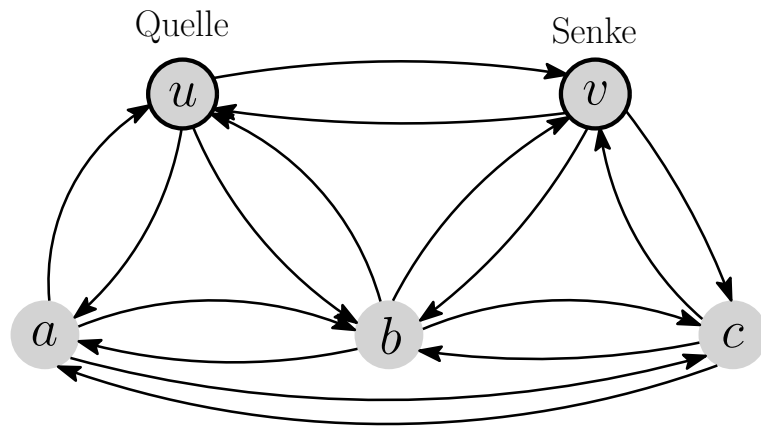
Sei $G = (V, E)$ ein Graph. G ist genau dann k-kantenzusammenhängend, wenn es für alle Paare von Knoten $u, v \in V$, $u \neq v$, mindestens k kantendisjunkte u-v-Pfade gibt.

Graph zu Netzwerk (für kantendisjunkte Pfade)

$$\overbrace{G = (V, E), u, v \in V}^{\text{Graph mit 2 Knoten}} \quad \mapsto \quad \overbrace{N_G^* = (V, A, c, u, v)}^{\text{Netzwerk}}$$

- $A := \{(x, y), (y, x) \mid \{x, y\} \in E\}$.
- $c \equiv 1$.



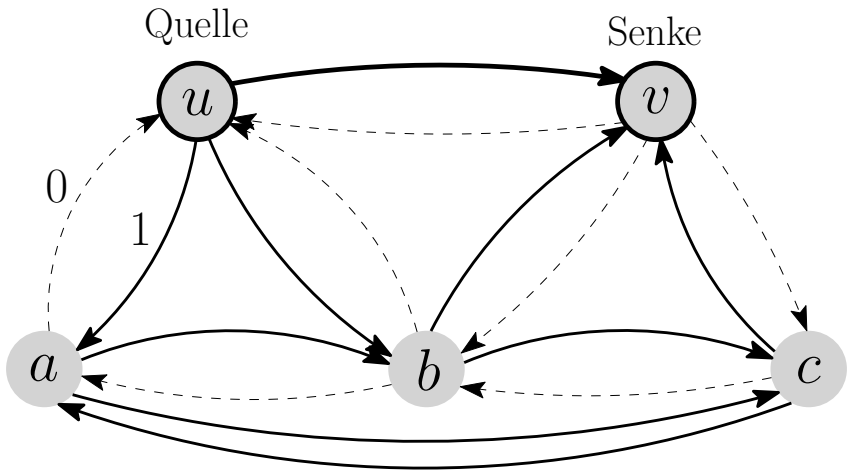


Nun haben wir entgegen gerichtete Kanten im Netzwerk!

Fluss zu kantendisjunkten Pfaden

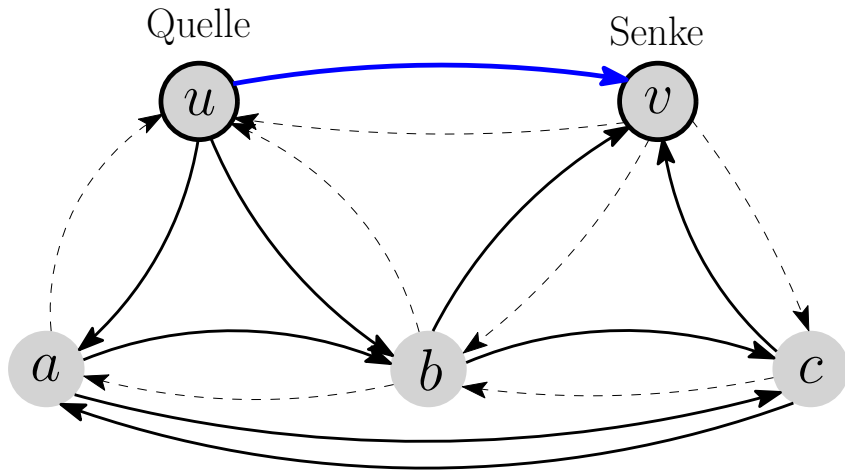
- Berechne ganzz. max. Fluss f in N_G^* $\Rightarrow$ Flusswerte $\in \{0, 1\}$.
- Für alle Knoten $w \notin \{u, v\}$ gilt: $\text{indeg}_f(w) = \text{outdeg}_f(w)$.
- $\text{val}(f) = \text{outdeg}_f(u) - \text{indeg}_f(u) = \text{indeg}_f(v) - \text{outdeg}_f(v)$.

Ein-/Ausgrade bzgl. Fluss 1 Kanten.



Fluss zu kantendisjunkten Pfaden

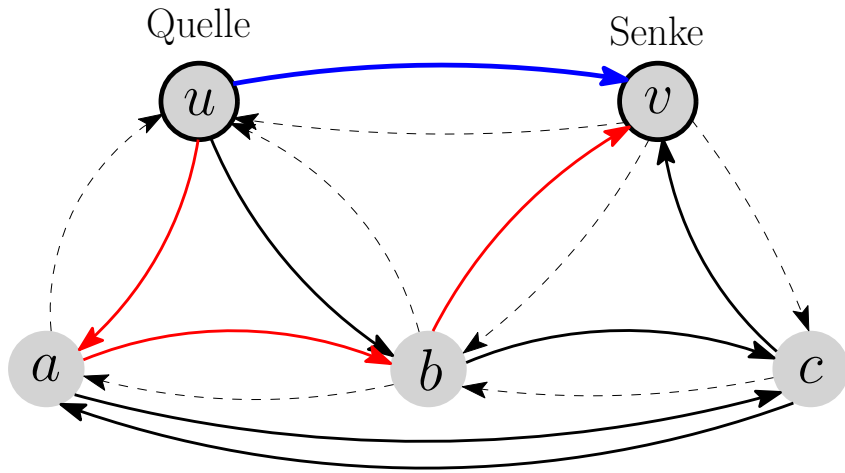
- Berechne ganzz. max. Fluss f in N_G^* $\Rightarrow$ Flusswerte $\in \{0, 1\}$.
- Für alle Knoten $w \notin \{u, v\}$ gilt: $\text{indeg}_f(w) = \text{outdeg}_f(w)$.
- $\text{val}(f) = \text{outdeg}_f(u) - \text{indeg}_f(u) = \text{indeg}_f(v) - \text{outdeg}_f(v)$.



- Beginnend bei u laufe entlang gerichteten ungebrauchten Kanten mit Fluss 1 bis man bei v ankommt. Unterwegs durchlaufene Kanten werden als gebraucht markiert.

Fluss zu kantendisjunkten Pfaden

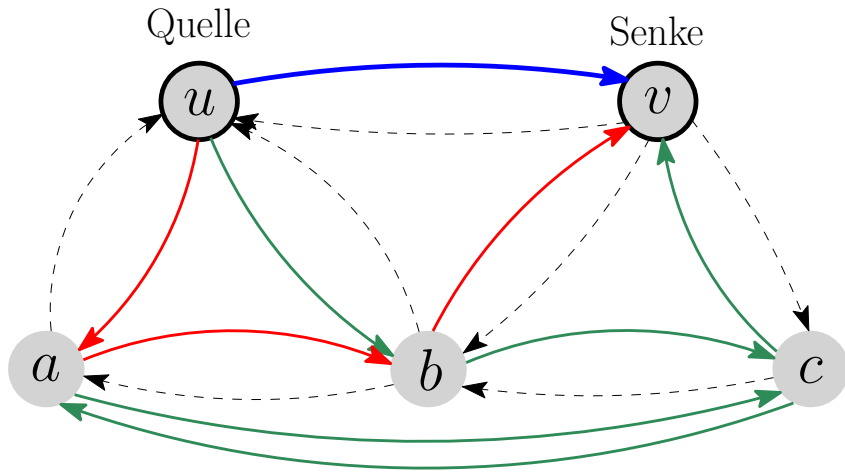
- Berechne ganzz. max. Fluss f in N_G^* $\Rightarrow$ Flusswerte $\in \{0, 1\}$.
- Für alle Knoten $w \notin \{u, v\}$ gilt: $\text{indeg}_f(w) = \text{outdeg}_f(w)$.
- $\text{val}(f) = \text{outdeg}_f(u) - \text{indeg}_f(u) = \text{indeg}_f(v) - \text{outdeg}_f(v)$.



- Beginnend bei u laufe entlang gerichteten ungebrauchten Kanten mit Fluss 1 bis man bei v ankommt. Unterwegs durchlaufene Kanten werden als gebraucht markiert.
- Wiederhole $\text{val}(f)$ Mal. Das gibt $\text{val}(f)$ kantendisjunkte Pfade

Fluss zu kantendisjunkten Pfaden

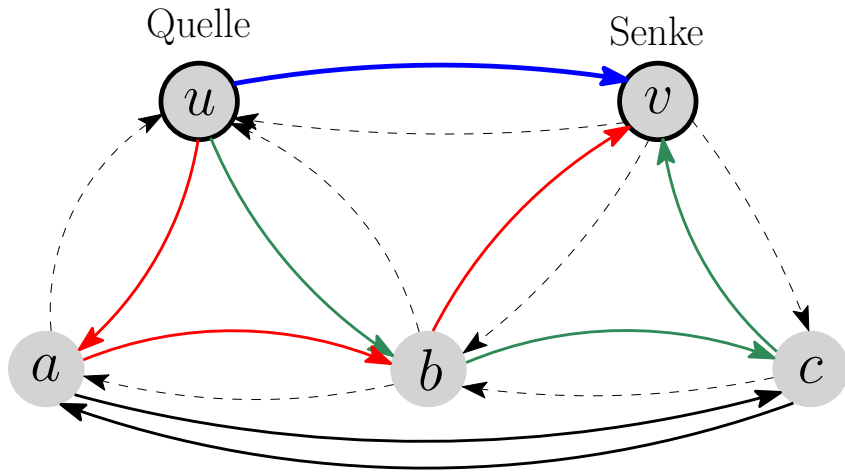
- Berechne ganzz. max. Fluss f in N_G^* $\Rightarrow$ Flusswerte $\in \{0, 1\}$.
- Für alle Knoten $w \notin \{u, v\}$ gilt: $\text{indeg}_f(w) = \text{outdeg}_f(w)$.
- $\text{val}(f) = \text{outdeg}_f(u) - \text{indeg}_f(u) = \text{indeg}_f(v) - \text{outdeg}_f(v)$.



- Beginnend bei u laufe entlang gerichteten ungebrauchten Kanten mit Fluss 1 bis man bei v ankommt. Unterwegs durchlaufene Kanten werden als gebraucht markiert.
- Wiederhole $\text{val}(f)$ Mal. Das gibt $\text{val}(f)$ kantendisjunkte Pfade (nach Entfernen von Kreisen).

Fluss zu kantendisjunkten Pfaden

- Berechne ganzz. max. Fluss f in N_G^* $\Rightarrow$ Flusswerte $\in \{0, 1\}$.
- Für alle Knoten $w \notin \{u, v\}$ gilt: $\text{indeg}_f(w) = \text{outdeg}_f(w)$.
- $\text{val}(f) = \text{outdeg}_f(u) - \text{indeg}_f(u) = \text{indeg}_f(v) - \text{outdeg}_f(v)$.



- Beginnend bei u laufe entlang gerichteten ungebrauchten Kanten mit Fluss 1 bis man bei v ankommt. Unterwegs durchlaufene Kanten werden als gebraucht markiert.
- Wiederhole $\text{val}(f)$ Mal. Das gibt $\text{val}(f)$ kantendisjunkte Pfade (nach Entfernen von Kreisen).

Maxflow-Mincut Theorem vs. Satz von Menger

Satz (**Maxflow-Mincut**)

Jedes Netzwerk erfüllt

$$\max_{f \text{ Fluss}} \mathsf{val}(f) = \min_{(S,T) \text{ s-t-Schnitt}} \mathsf{cap}(S,T) \ .$$

Und ganzzahlige Netzwerke haben ganzzahlige maximale Flüsse.

Maxflow-Mincut Theorem vs. Satz von Menger

Satz (**Maxflow-Mincut**)

Jedes Netzwerk erfüllt

$$\max_{f \text{ Fluss}} \mathrm{val}(f) = \min_{(S,T) \text{ s-t-Schnitt}} \mathrm{cap}(S,T) \ .$$

Und ganzzahlige Netzwerke haben ganzzahlige maximale Flüsse.

$$\Downarrow$$

Satz (**Menger**, Variante)

Sei G ein Graph mit Knoten u und v, $u \neq v$.

$$\begin{gathered} \max \#\ \textit{kantendisjunkter } u\textit{-}v\textit{-Pfade in } G \\ = \\ \min \#\ \textit{Kanten, die } u \textit{ und } v \textit{ trennen} \end{gathered}$$

(„trennen“ heisst, nach Entfernen der Kanten sind u und v in verschiedenen Zusammenhangskomponenten des Graphen).

Maxflow-Mincut Theorem vs. Satz von Menger

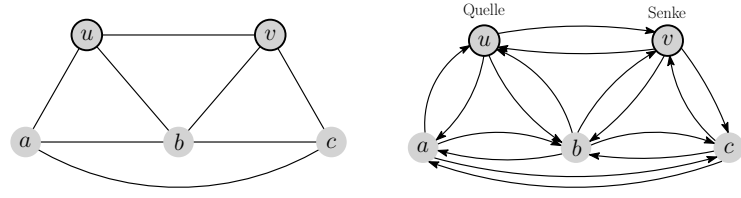
Satz (**Maxflow-Mincut**)

Jedes Netzwerk erfüllt

$$\max_{f \text{ Fluss}} \mathrm{val}(f) = \min_{(S,T) \text{ s-t-Schnitt}} \mathrm{cap}(S,T) \ .$$

Und ganzzahlige Netzwerke haben ganzzahlige maximale Flüsse.

$\Downarrow$



Satz (**Menger**, Variante)

Sei G ein Graph mit Knoten u und v, $u \neq v$.

$$\max \ \#\ \textit{kantendisjunkter u-v-Pfade in } G$$
$$=$$
$$\min \ \#\ \textit{Kanten, die } u \textit{ und } v \textit{ trennen}$$

(„trennen“ heisst, nach Entfernen der Kanten sind u und v in verschiedenen Zusammenhangskomponenten des Graphen).

Flüsse helfen bei ...

- Matchings, hier: bipartites maximum Matching in $O(mn)$ (geht besser in $O((m+n)\sqrt{n})$ [Hopcroft&Karp'73]).
- Schnitten zwischen Knoten u und v (Knoten-/Kantenzusammenhang).
- Kantendisjunkten Pfaden (auch knotendisjunkten Pfaden).
- Beweis Satz von Menger (aus Maxflow-Mincut).

Flüsse in Netzwerken: Anwendungen (Teil 2)

Kopf CT [CT-MRTinstitut Berlin]

III Bildsegmentierung

Bildsegmentierung. Gegeben ein Bild (aus Pixeln mit Farbwerten), trenne Vordergrund von Hintergrund.

Intuitiv: Ein Bild ist ein Menge P von Pixeln (z.B. gitterförmig angeordnet) mit Farben (z.B. RGB, Grauwerte), mit einer Nachbarschaftsrelation, die sagt welche Pixel nebeneinander liegen.

Modellierung

Ein **Bild** ist ein Graph (P, E) mit Farbinformation $\chi\colon P \to$ Farben.

Modellierung

Ein **Bild** ist ein Graph (P, E) mit Farbinformation $\chi\colon P \to$ Farben.

Jemand extrahiert aus den Farben der Pixel individuell eine Einschätzung, ob das Pixel im Vordergrund oder Hintergrund liegt:

$$\alpha : P \to \mathbb{R}_0^+ \qquad \alpha_p \text{ grösser } \Rightarrow \text{ eher im Vordergrund}$$
$$\beta : P \to \mathbb{R}_0^+ \qquad \beta_p \text{ grösser } \Rightarrow \text{ eher im Hintergrund}$$

Modellierung

Ein **Bild** ist ein Graph (P, E) mit Farbinformation $\chi\colon P \to$ Farben.

Jemand extrahiert aus den Farben der Pixel individuell eine Einschätzung, ob das Pixel im Vordergrund oder Hintergrund liegt:

$$\begin{array}{lll} \alpha : P \to \mathbb{R}_0^+ & \alpha_p \text{ grösser } \Rightarrow & \text{eher im Vordergrund} \\ \beta : P \to \mathbb{R}_0^+ & \beta_p \text{ grösser } \Rightarrow & \text{eher im Hintergrund} \end{array}$$

Erster Ansatz:

$$\begin{array}{lll} \text{Vordergrund } A & := & \{p \in P \mid \alpha_p > \beta_p\} \text{ und} \\ \text{Hintergrund } B & := & P \setminus A\,. \end{array}$$

Nachteil: Die Aufteilung wir in vielen Fällen zu feinkörnig.

Modellierung

Ein **Bild** ist ein Graph (P, E) mit Farbinformation $\chi : P \to$ Farben.

Wir erhalten eine dritte Einschätzung, ob benachbarte Pixel eher im gleichen Teil (Vorder-/Hintergund) liegen.

$$\alpha : P \to \mathbb{R}_0^+ \qquad \alpha_p \text{ grösser } \Rightarrow \text{ eher im Vordergrund}$$
$$\beta : P \to \mathbb{R}_0^+ \qquad \beta_p \text{ grösser } \Rightarrow \text{ eher im Hintergrund}$$
$$\gamma : E \to \mathbb{R}_0^+ \qquad \gamma_e \text{ grösser } \Rightarrow \text{ eher im gleichen Teil}$$

Modellierung

Ein **Bild** ist ein Graph (P, E) mit Farbinformation $\chi\colon P \to$ Farben.

Wir erhalten eine dritte Einschätzung, ob benachbarte Pixel eher im gleichen Teil (Vorder-/Hintergund) liegen.

$$\begin{array}{lll} \alpha : P \to \mathbb{R}_0^+ & \alpha_p \text{ grösser} \Rightarrow & \text{eher im Vordergrund} \\ \beta : P \to \mathbb{R}_0^+ & \beta_p \text{ grösser} \Rightarrow & \text{eher im Hintergrund} \\ \gamma : E \to \mathbb{R}_0^+ & \gamma_e \text{ grösser} \Rightarrow & \text{eher im gleichen Teil} \end{array}$$

Qualitätsfunktion für Vorder-/Hintergrundspartition (A, B) von P:

$$q(A, B) := \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p - \sum_{e \in E,\, |e \cap A| = 1} \gamma_e \ .$$

Unsere Problemstellung

Bildsegmentierung. Gegeben ein Bild (P, E) mit

$$\alpha : P \to \mathbb{R}_0^+, \quad \beta : P \to \mathbb{R}_0^+, \quad \gamma : E \to \mathbb{R}_0^+ \ ,$$

finde eine Partition (A, B) von P die

$$q(A, B) := \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p - \sum_{e \in E,\, |e \cap A| = 1} \gamma_e \ .$$

maximiert.

Umformung von $q(A, B)$

$$q(A, B) := \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p - \sum_{e \in E,\, |e \cap A| = 1} \gamma_e \ .$$

Mit $Q := \sum_{p \in P} (\alpha_p + \beta_p)$ gilt

$$q(A, B) = Q - \sum_{p \in A} \beta_p - \sum_{p \in B} \alpha_p - \sum_{e \in E, |e \cap A| = 1} \gamma_e \ .$$

Umformung von $q(A, B)$

$$q(A, B) := \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p - \sum_{e \in E,\, |e \cap A| = 1} \gamma_e \ .$$

Mit $Q := \sum_{p \in P} (\alpha_p + \beta_p)$ gilt

$$q(A, B) = Q - \sum_{p \in A} \beta_p - \sum_{p \in B} \alpha_p - \sum_{e \in E, |e \cap A| = 1} \gamma_e \ .$$

$q(A, B)$ zu maximieren is äquivalent zur Minimierung von

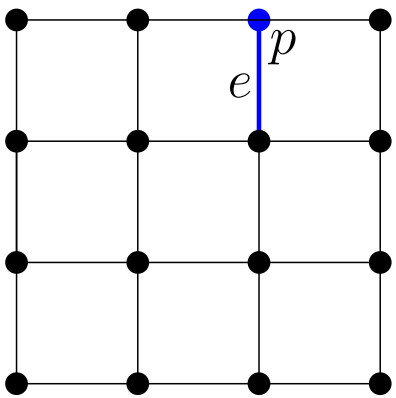
$$q'(A, B) := \sum_{p \in A} \beta_p + \sum_{p \in B} \alpha_p + \sum_{e \in E, |e \cap A| = 1} \gamma_e \ .$$

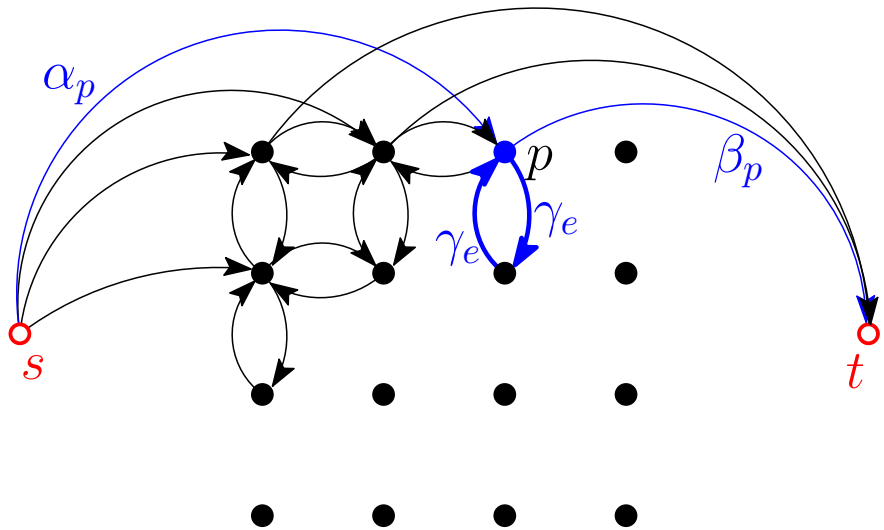
Vom Bild zum Netzwerk

$N := (P \cup \{s, t\}, \vec{E}, c, s, t)$:

- Neue Knoten s und t, Quelle und Senke im Netzwerk.
- Die Quelle s hat eine gerichtete Kante zu jedem Pixel $p \in P$ mit Kapazität α_p.
- Jedes Pixel p hat eine gerichtete Kante zur Senke t mit Kapazität β_p.
- Für jede Kante $e = \{p, p'\} \in E$ gibt es zwei gerichtete Kanten (p, p') und (p', p), je mit Kapazität γ_e.

Bild von „Vom Bild zum Netzwerk“





Kapazität eines s-t-Schnitts in N

Sei (S, T) ein s-t-Schnitt, und $A := S \setminus \{s\}$ und $B := T \setminus \{t\}$.
Welche Kanten mit welchem Beitrag sind in diesem s-t-Schnitt?

- Kanten (s, p) mit $p \in B$; Beitrag zu $\mathrm{cap}(S, T)$ ist $\sum_{p \in B} \alpha_p$.
- Kanten (p, t) mit $p \in A$; Beitrag zu $\mathrm{cap}(S, T)$ ist $\sum_{p \in A} \beta_p$.
- Kanten (p, p') des Netzwerks in $A \times B$ mit Beitrag

$$\sum_{(p,p') \in A \times B, \{p,p'\} \in E} \gamma_{(p,p')}.$$

Es folgt

$$\begin{aligned} q'(A, B) &:= \sum_{p \in A} \beta_p + \sum_{p \in B} \alpha_p + \sum_{e \in E, |e \cap A| = 1} \gamma_e \\ &= \mathrm{cap}(S, T) \end{aligned}$$

Anmerkungen

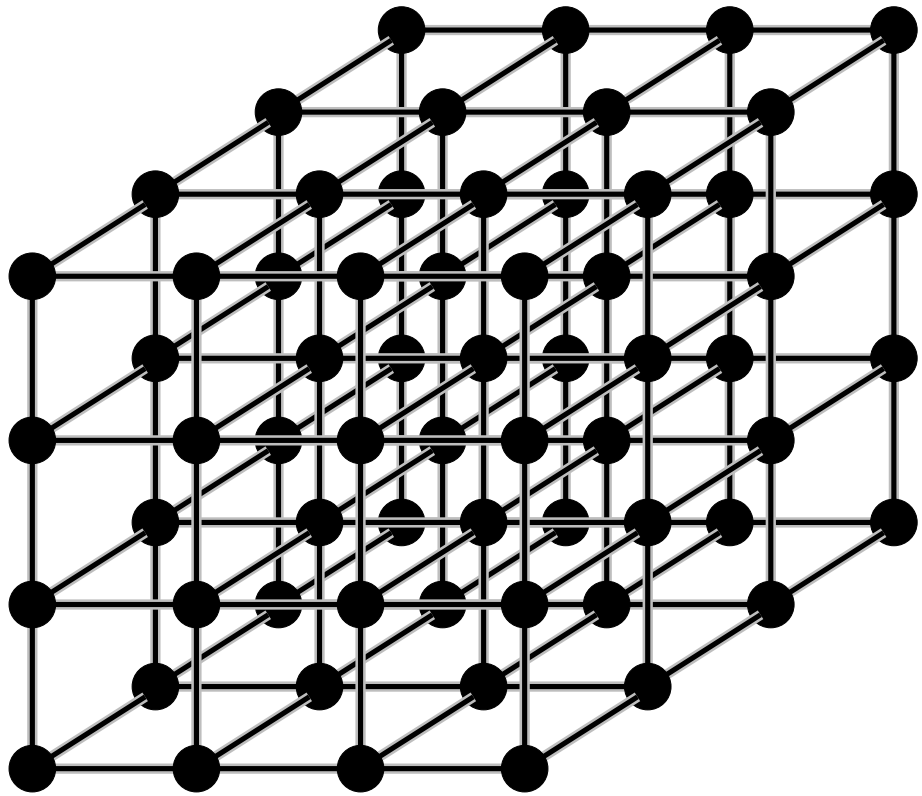
- Die optimale Partition (A, B) kann also mit Hilfe von MaxFlow für den minimalen s-t-Schnitt berechnet werden.

Anmerkungen

- Die optimale Partition (A, B) kann also mit Hilfe von MaxFlow für den minimalen s-t-Schnitt berechnet werden.

Dieser Lösungsansatz kommt in Anwendungen zum Einsatz,

- insbesondere auch für höherdimensionale Bilder (mit Voxeln), z.B. Computertomographie.



Minimale Schnitte in Graphen: Einführung und erster Ansatz (Teil 1)

Problemstellung

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme die Kardinalität eines *minimalen Kantenschnitts*.

Problemstellung

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme die Kardinalität eines *minimalen Kantenschnitts*.

Multigraph: Ungerichteter, ungewichteter Graph $G = (V, E)$ ohne Schleifen, aber möglicherweise mit mehreren Kanten zwischen demselben Knotenpaar. (Kann auch durch positiv ganzzahlige Kantengewichte realisiert werden.)

Problemstellung

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme die Kardinalität eines *minimalen Kantenschnitts*.

Multigraph: Ungerichteter, ungewichteter Graph $G = (V, E)$ ohne Schleifen, aber möglicherweise mit mehreren Kanten zwischen demselben Knotenpaar. (Kann auch durch positiv ganzzahlige Kantengewichte realisiert werden.)

Kantenschnitt in einem Multigraph $G = (V, E)$: Menge von Kanten C, so dass $(V, E \setminus C)$ unzusammenhängend ist.

Problemstellung

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme die Kardinalität eines *minimalen Kantenschnitts*.

Multigraph: Ungerichteter, ungewichteter Graph $G = (V, E)$ ohne Schleifen, aber möglicherweise mit mehreren Kanten zwischen demselben Knotenpaar. (Kann auch durch positiv ganzzahlige Kantengewichte realisiert werden.)

Kantenschnitt in einem Multigraph $G = (V, E)$: Menge von Kanten C, so dass $(V, E \setminus C)$ unzusammenhängend ist.

(Kantenmenge $C \subseteq E$ vs. Partition (S, T) von V)

Problemstellung

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme die Kardinalität eines *minimalen Kantenschnitts*.

Multigraph: Ungerichteter, ungewichteter Graph $G = (V, E)$ ohne Schleifen, aber möglicherweise mit mehreren Kanten zwischen demselben Knotenpaar. (Kann auch durch positiv ganzzahlige Kantengewichte realisiert werden.)

Kantenschnitt in einem Multigraph $G = (V, E)$: Menge von Kanten C, so dass $(V, E \setminus C)$ unzusammenhängend ist.

(Kantenmenge $C \subseteq E$ vs. Partition (S, T) von V)

Mit $\mu(G)$ bezeichnen wir die Kardinalität eines kleinstmöglichen Kantenschnitts in G, d.h.

$$\mu(G) := \min_{C \subseteq E, (V, E \setminus C) \text{ unzusammenhängend}} |C|$$

Beispiele

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme $\mu(G)$.

Beispiele

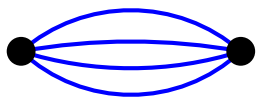
MIN-CUT Problem. Gegeben ein Multigraph G, bestimme $\mu(G)$.

Für G nicht zusammenhängend gilt $\mu(G) = 0$.

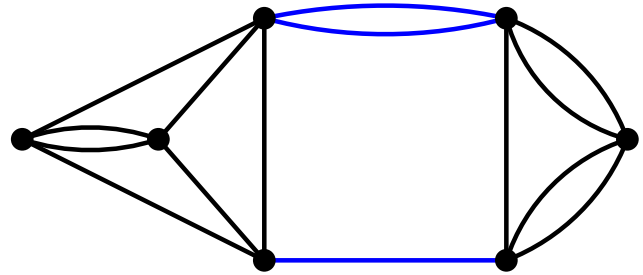
Beispiele

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme $\mu(G)$.

Für G nicht zusammenhängend gilt $\mu(G) = 0$.



$\mu = 4$

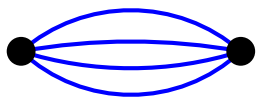


$\mu = 3$

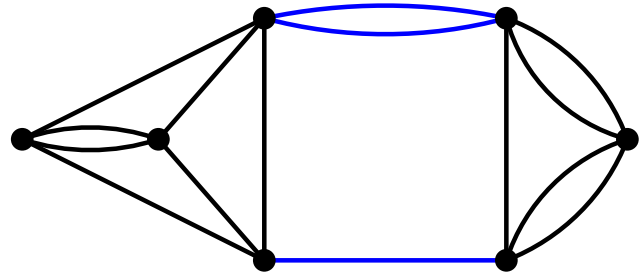
Beispiele

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme $\mu(G)$.

Für G nicht zusammenhängend gilt $\mu(G) = 0$.



$\mu = 4$



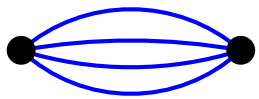
$\mu = 3$

In einem Multigraph $G = (V, E)$ ist der **Grad** $\deg(v) = \deg_G(v)$ eines Knoten v die #inzidenter Kanten (nicht die #Nachbarn).

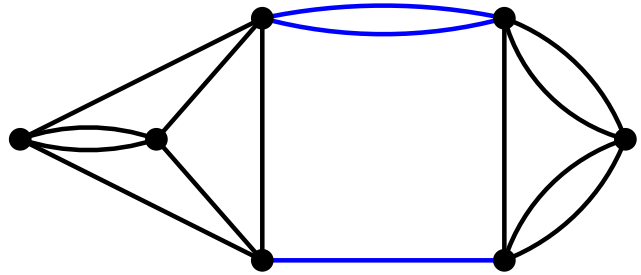
Beispiele

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme $\mu(G)$.

Für G nicht zusammenhängend gilt $\mu(G) = 0$.



$\mu = 4$



$\mu = 3$

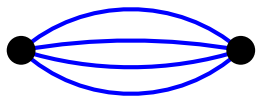
In einem Multigraph $G = (V, E)$ ist der **Grad** $\deg(v) = \deg_G(v)$ eines Knoten v die #inzidenter Kanten (nicht die #Nachbarn). So gilt

$$|E| = \frac{1}{2} \sum_{v \in V} \deg(v) \quad \text{und} \quad \mu(G) \leq \min_{v \in V} \deg(v) \ .$$

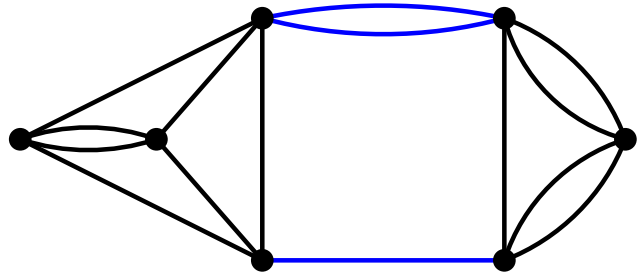
Beispiele

MIN-CUT Problem. Gegeben ein Multigraph G, bestimme $\mu(G)$.

Für G nicht zusammenhängend gilt $\mu(G) = 0$.



$\mu = 4$
$\min \deg = 4$

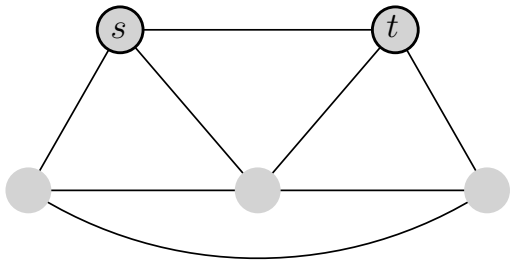


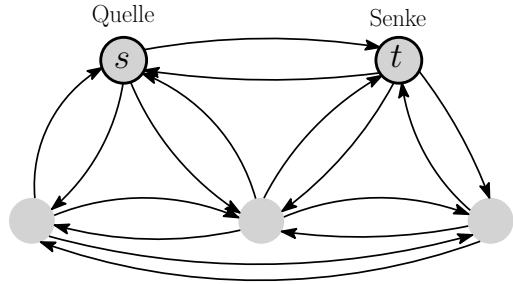
$\mu = 3$, $\min \deg = 4$

In einem Multigraph $G = (V, E)$ ist der **Grad** $\deg(v) = \deg_G(v)$ eines Knoten v die #inzidenter Kanten (nicht die #Nachbarn). So gilt

$$|E| = \frac{1}{2} \sum_{v \in V} \deg(v) \quad \text{und} \quad \mu(G) \leq \min_{v \in V} \deg(v) \ .$$

Erste bekannte Lösung

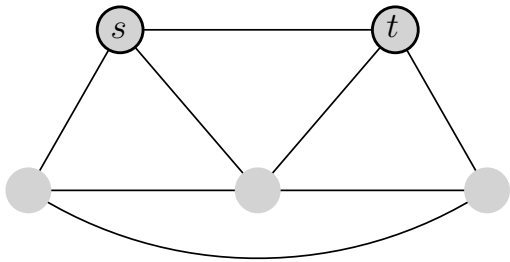


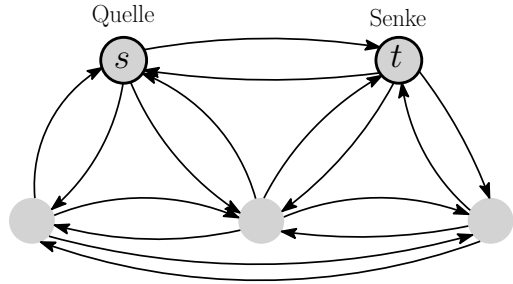


Wir können bereits den kleinsten s-t-Schnitt berechnen, d.h. die kleinste #Kanten, die man entfernen muss, um s von t zu trennen.

Zeit: $O(mnU)$, bzw. $O(mn(1 + \log U))$ oder $O(mn \log n)$.

Erste bekannte Lösung



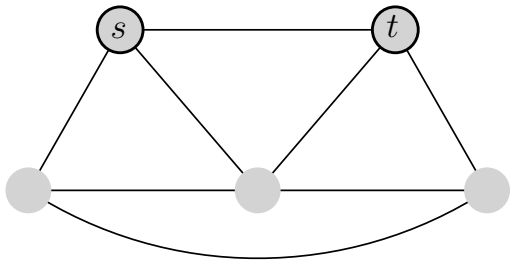


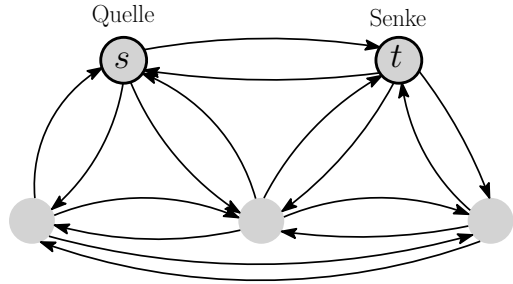
Wir können bereits den kleinsten s-t-Schnitt berechnen, d.h. die kleinste #Kanten, die man entfernen muss, um s von t zu trennen.

Zeit: $O(mnU)$, bzw. $O(mn(1 + \log U))$ oder $O(mn \log n)$.

Für unser Problem fixieren wir ein s und betrachten alle $t \in V \setminus \{s\}$. Jeder Schnitt „ist“ ein s-t-Schnitt, für ein $t \in V \setminus \{s\}$.

Erste bekannte Lösung





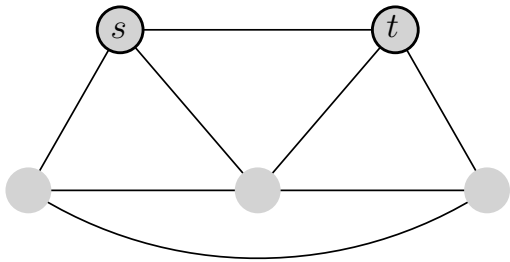
Wir können bereits den kleinsten s-t-Schnitt berechnen, d.h. die kleinste #Kanten, die man entfernen muss, um s von t zu trennen.

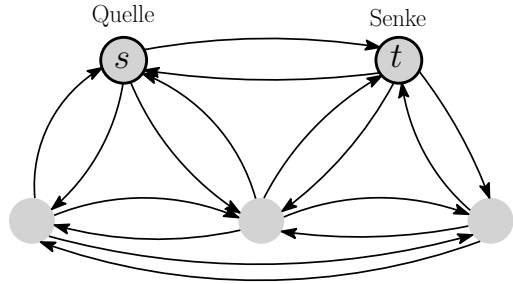
Zeit: $O(mnU)$, bzw. $O(mn(1 + \log U))$ oder $O(mn \log n)$.

Für unser Problem fixieren wir ein s und betrachten alle $t \in V \setminus \{s\}$. Jeder Schnitt „ist“ ein s-t-Schnitt, für ein $t \in V \setminus \{s\}$.

$(n - 1)$ Mal $O(mn \log n)$, gibt $O(mn^2 \log n) = O(n^4 \log n)$.

Erste bekannte Lösung





Wir können bereits den kleinsten s-t-Schnitt berechnen, d.h. die kleinste #Kanten, die man entfernen muss, um s von t zu trennen.

Zeit: $O(mnU)$, bzw. $O(mn(1 + \log U))$ oder $O(mn \log n)$.

Für unser Problem fixieren wir ein s und betrachten alle $t \in V \setminus \{s\}$. Jeder Schnitt „ist“ ein s-t-Schnitt, für ein $t \in V \setminus \{s\}$.

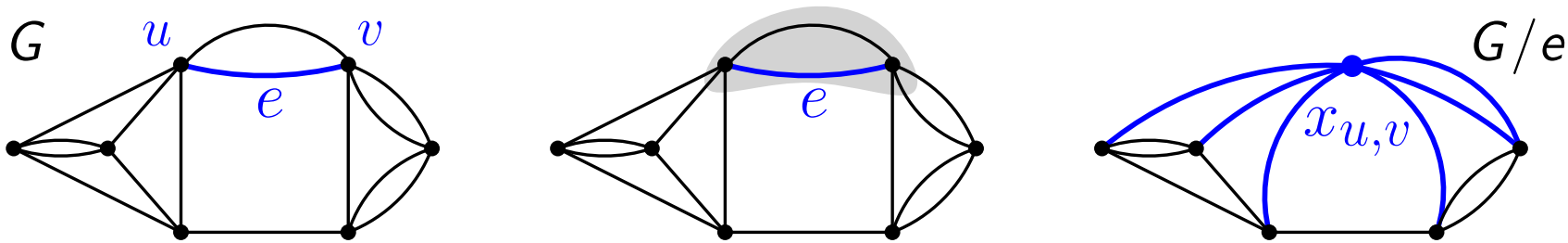
$(n-1)$ Mal $O(mn \log n)$, gibt $O(mn^2 \log n) = O(n^4 \log n)$.

$O(n^4 \log n)$ ist unsere Messlatte, die wir unterbieten wollen.

Neuer Ansatz – mittels Kantenkontraktion

Gegeben $G = (V, E)$, $e = \{u, v\} \in E$.

Kontraktion von e: Verschmilzt u und v zu einem neuen Knoten $x_{u,v}$, der nun zu allen Kanten inzident ist, zu denen u oder v inzident war. Die Kanten zwischen u und v verschwinden.

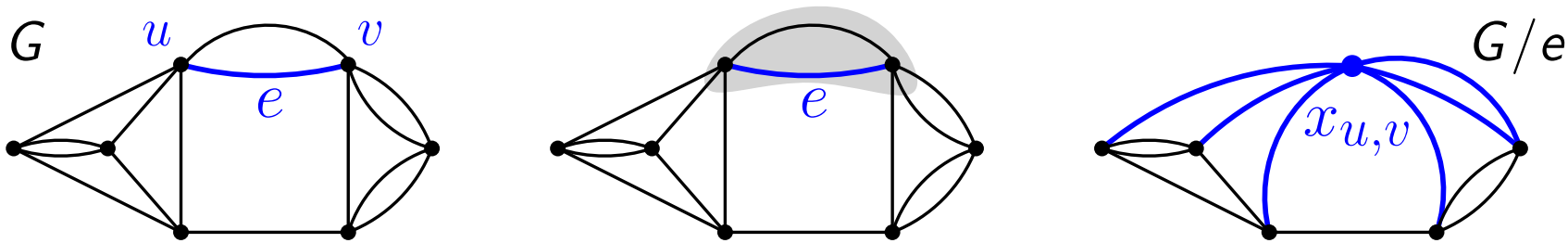


Den entstehenden Graph bezeichnen wir mit G/e.

Neuer Ansatz – mittels Kantenkontraktion

Gegeben $G = (V, E)$, $e = \{u, v\} \in E$.

Kontraktion von e: Verschmilzt u und v zu einem neuen Knoten $x_{u,v}$, der nun zu allen Kanten inzident ist, zu denen u oder v inzident war. Die Kanten zwischen u und v verschwinden.



Den entstehenden Graph bezeichnen wir mit G/e.

Für $k :=$ #Kanten zwischen u und v gilt

$$\deg_{G/e} x_{u,v} = \deg_G(u) + \deg_G(v) - 2k \ .$$

und $|E(G/e)| = |E(G)| - k$.

Kontraktion erhält die meisten Schnitte

Sei $e = \{u, v\}$. Es gibt eine natürliche Bijektion

$$E(G) \text{ ohne Kanten zw. } u \text{ und } v \quad \longrightarrow \quad E(G/e) \ .$$

Für $w, w' \in V(G) \setminus \{u, v\}$:

$$\{w, w'\} \mapsto \{w, w'\}, \ \{w, u\} \mapsto \{w, x_{u,v}\}, \ \{w, v\} \mapsto \{w, x_{u,v}\}$$

Kontraktion erhält die meisten Schnitte

Sei $e = \{u, v\}$. Es gibt eine natürliche Bijektion

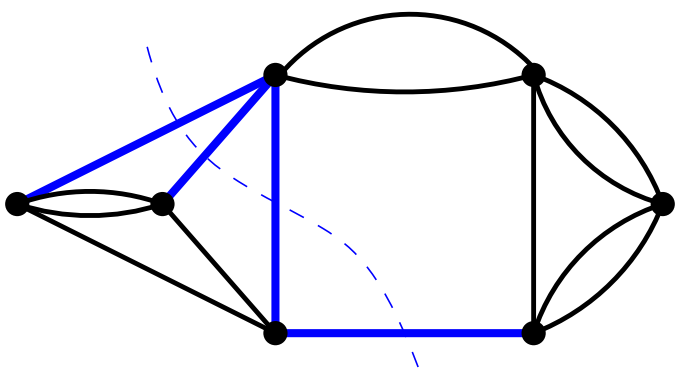
$$E(G) \text{ ohne Kanten zw. } u \text{ und } v \quad \longrightarrow \quad E(G/e) \ .$$

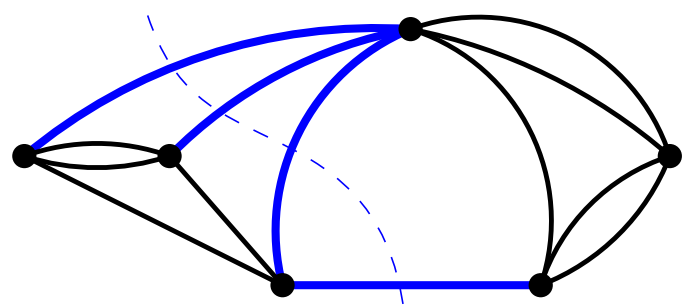
Für $w, w' \in V(G) \setminus \{u, v\}$:

$$\{w, w'\} \mapsto \{w, w'\}, \ \{w, u\} \mapsto \{w, x_{u,v}\}, \ \{w, v\} \mapsto \{w, x_{u,v}\}$$

Dies induziert eine Bijektion

$$\text{Schnitte in } G \text{ ohne } e \quad \longrightarrow \quad \text{alle Schnitte in } G/e$$





Minimaler Schnitt in G vs. G/e

Lemma

Sei $G = (V, E)$ ein Multigraph, $e \in E$. Dann gilt

$$\mu(G/e) \geq \mu(G) \ .$$

Falls G einen minimalen Schnitt C mit $e \notin C$ hat, dann gilt

$$\mu(G/e) = \mu(G) \ .$$

Minimaler Schnitt in G vs. G/e

Lemma
Sei $G = (V, E)$ ein Multigraph, $e \in E$. Dann gilt

$$\mu(G/e) \geq \mu(G) \ .$$

Falls G einen minimalen Schnitt C mit $e \notin C$ hat, dann gilt

$$\mu(G/e) = \mu(G) \ .$$

μ kann durch Kontraktion einer Kante e nie fallen.
μ bleibt gleich, falls es einen minimalen Schnitt ohne e gibt.

Minimaler Schnitt in G vs. G/e

Lemma
Sei $G = (V, E)$ ein Multigraph, $e \in E$. Dann gilt

$$\mu(G/e) \geq \mu(G) \ .$$

Falls G einen minimalen Schnitt C mit $e \notin C$ hat, dann gilt

$$\mu(G/e) = \mu(G) \ .$$

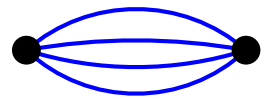
μ kann durch Kontraktion einer Kante e nie fallen.
μ bleibt gleich, falls es einen minimalen Schnitt ohne e gibt.

Wie finden wir eine Kante e die μ erhält?

Auf gut Glück – zufällige Kantenkontraktionen

Cut(G) — G zusammenhängender Multigraph

1: $G' \leftarrow G$
2: **while** $|V(G')| > 2$ **do**
3: $\quad e \leftarrow$ gleichverteilt zufällige Kante in G'
4: $\quad G' \leftarrow G'/e$
5: **return** Grösse des eindeutigen Schnitts in G'



Sei $n := |V(G)|$. Wir setzen voraus:

- Kantenkontraktion in $O(n)$ Zeit.
- Gleichverteilt zufällige Kante in G in $O(n)$ Zeit.

(Erfordert Darstellung der Mehrfachkanten durch Kantengewichte.)

Damit kann man Cut(G) mit Laufzeit $O(n^2)$ implementieren.

Aber: Was können wir mit dem berechneten Wert anfangen?

Erhalt von μ

Zur Erinnerung:

μ kann durch Kontraktion einer Kante e nie fallen.
μ bleibt gleich, falls es einen minimalen Schnitt ohne e gibt.

Erhalt von μ

Zur Erinnerung:

μ kann durch Kontraktion einer Kante e nie fallen.
μ bleibt gleich, falls es einen minimalen Schnitt ohne e gibt.

- Ergebnis des Algorithmus ist nie kleiner als $\mu(G)$.
- Falls es einen minimalen Schnitt C gibt, aus dem nie eine Kante kontrahiert wird, so gibt der Algorithmus $\mu(G)$ aus.

Erhalt von μ

Zur Erinnerung:

μ kann durch Kontraktion einer Kante e nie fallen.
μ bleibt gleich, falls es einen minimalen Schnitt ohne e gibt.

- Ergebnis des Algorithmus ist nie kleiner als $\mu(G)$.
- Falls es einen minimalen Schnitt C gibt, aus dem nie eine Kante kontrahiert wird, so gibt der Algorithmus $\mu(G)$ aus.

Lemma
Sei $G = (V, E)$, $n := |V|$. Für e gleichverteilt zufällig in E gilt

$$\Pr\left[\mu(G) = \mu(G/e)\right] \geq 1 - \frac{2}{n} .$$

Erhalt des minimalen Schnitts

Lemma

Sei $G = (V, E)$, $n := |V|$. Für Kante e gleichverteilt zufällig unter den Kanten in G gilt $\Pr[\mu(G) = \mu(G/e)] \geq 1 - \frac{2}{n}$.

Beweis. Sei C ein minimaler Schnitt in G und $k := |C| = \mu(G)$.

Erhalt des minimalen Schnitts

Lemma

Sei $G = (V, E)$, $n := |V|$. Für Kante e gleichverteilt zufällig unter den Kanten in G gilt $\Pr[\mu(G) = \mu(G/e)] \geq 1 - \frac{2}{n}$.

Beweis. Sei C ein minimaler Schnitt in G und $k := |C| = \mu(G)$.

Wir wissen, dass $\forall v \in V : \deg_G(v) \geq k$ und daher gilt

$$|E| = \frac{1}{2} \sum_{v \in V} \deg_G(v) \geq \frac{kn}{2} .$$

Erhalt des minimalen Schnitts

Lemma

Sei $G = (V, E)$, $n := |V|$. Für Kante e gleichverteilt zufällig unter den Kanten in G gilt $\Pr[\mu(G) = \mu(G/e)] \geq 1 - \frac{2}{n}$.

Beweis. Sei C ein minimaler Schnitt in G und $k := |C| = \mu(G)$.

Wir wissen, dass $\forall v \in V : \deg_G(v) \geq k$ und daher gilt

$$|E| = \frac{1}{2} \sum_{v \in V} \deg_G(v) \geq \frac{kn}{2} .$$

Wegen „$e \notin C \Rightarrow \mu(G/e) = \mu(G)$“ gilt

$$\begin{aligned} \Pr[\mu(G) = \mu(G/e)] &\geq \Pr[e \notin C] = 1 - \frac{|C|}{|E|} \\ &\geq 1 - \frac{k}{kn/2} = 1 - \frac{2}{n} . \end{aligned}$$

Erfolgswahrscheinlichkeit

$\hat{p}(G) :=$ Wahrscheinlichkeit, dass Cut(G) den Wert $\mu(G)$ ausgibt

$$\hat{p}(n) := \inf_{G=(V,E),\,|V|=n} \hat{p}(G) \ .$$

Lemma

Es gilt für alle $n \geq 3$

$$\hat{p}(n) \geq \left(1 - \frac{2}{n}\right) \cdot \hat{p}(n-1) \ .$$

Erfolgswahrscheinlichkeit

Lemma

Für alle $n \geq 3$ gilt $\hat{p}(n) \geq (1 - \frac{2}{n}) \cdot \hat{p}(n-1)$.

Beweis. Sei $G = (V, E)$, $n := |V|$. Damit Cut(G) tatsächlich $\mu(G)$ ausgibt, müssen die beiden folgenden Ereignisse eintreten:

- $E_1 :=$ Ereignis $\mu(G) = \mu(G/e)$.
- $E_2 :=$ Ereignis, dass Cut(G/e) den Wert $\mu(G/e)$ ausgibt.

Es gilt nun

$$\hat{p}(G) = \Pr[E_1 \wedge E_2] = \Pr[E_1] \cdot \Pr[E_2 \mid E_1] \geq (1 - 2/n) \cdot \hat{p}(n-1) \ .$$

Da dies für jeden Multigraph G mit n Knoten gilt, folgt auch $\hat{p}(n) \geq (1 - \frac{2}{n}) \cdot \hat{p}(n-1)$.

Erfolgswahrscheinlichkeit

Es gilt also $\hat{p}(n) \geq \frac{n-2}{n} \cdot \hat{p}(n-1)$. Wir erhalten so

$$\hat{p}(n) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} \cdot \underbrace{\hat{p}(2)}_{=1} = \frac{2}{n(n-1)}$$

Lemma

Für alle $n \geq 2$ gilt

$$\hat{p}(n) \geq \frac{2}{n(n-1)} = 1/\binom{n}{2}$$

Das heisst, der Erwartungswert der #Wiederholungen, bis wir das erste Mal $\mu(G)$ ausgeben ist höchstens $\binom{n}{2}$.

Erstes Ergebnis

Wir wiederholen den Algorithmus Cut(G) $\lambda\binom{n}{2}$ mal, für ein $\lambda > 0$, und geben dann kleinsten je erhaltenen Wert aus.

Zur Erinnerung: Ein Aufruf von Cut(G) benötigt Zeit $O(n^2)$.

Erstes Ergebnis

Wir wiederholen den Algorithmus Cut(G) $\lambda\binom{n}{2}$ mal, für ein $\lambda > 0$, und geben dann kleinsten je erhaltenen Wert aus.

Zur Erinnerung: Ein Aufruf von Cut(G) benötigt Zeit $O(n^2)$.

Satz

Für den Algorithmus der $\lambda\binom{n}{2}$-maligen Wiederholung von Cut(G) *gilt:*

(1) *Der Algorithmus hat eine Laufzeit von $O(\lambda n^4)$.*

(2) *Der kleinste angetroffene Wert ist mit einer Wahrscheinlichkeit von mindestens $1 - e^{-\lambda}$ gleich $\mu(G)$.*

Erstes Ergebnis

Wir wiederholen den Algorithmus Cut(G) $\lambda\binom{n}{2}$ mal, für ein $\lambda > 0$, und geben dann kleinsten je erhaltenen Wert aus.

Zur Erinnerung: Ein Aufruf von Cut(G) benötigt Zeit $O(n^2)$.

Satz

Für den Algorithmus der $\lambda\binom{n}{2}$-maligen Wiederholung von Cut(G) *gilt:*

(1) *Der Algorithmus hat eine Laufzeit von $O(\lambda n^4)$.*

(2) *Der kleinste angetroffene Wert ist mit einer Wahrscheinlichkeit von mindestens $1 - e^{-\lambda}$ gleich $\mu(G)$.*

Mit $\lambda := \ln n$, haben wir **Zeit** $O(n^4 \log n)$ mit **Fehlerw'keit** $\leq 1/n$.

Die Laufzeit hatten wir aber schon deterministisch (ohne Fehler)!

Siehe Teil 2.

Minimale Schnitte in Graphen: Bootstrapping (Teil 2)

$$
\begin{array}{c}
n^2 \\
\uparrow \\
\vdots \\
\uparrow \\
n^{2.67} \\
\uparrow \\
n^3 \\
\uparrow \\
n^4 \\
\uparrow
\end{array}
$$

Unser erstes Ergebnis

Satz

Für den Algorithmus der $\lambda\binom{n}{2}$-maligen Wiederholung von $\mathrm{Cut}(G)$ *gilt:*

(1) *Der Algorithmus hat eine Laufzeit von* $O(\lambda n^4)$.

(2) *Der kleinste angetroffene Wert ist mit einer Wahrscheinlichkeit von mindestens* $1 - e^{-\lambda}$ *gleich* $\mu(G)$.

Unser erstes Ergebnis

Satz

Für den Algorithmus der $\lambda\binom{n}{2}$-maligen Wiederholung von $\mathrm{Cut}(G)$ *gilt:*

(1) *Der Algorithmus hat eine Laufzeit von $O(\lambda n^4)$.*

(2) *Der kleinste angetroffene Wert ist mit einer Wahrscheinlichkeit von mindestens $1 - e^{-\lambda}$ gleich $\mu(G)$.*

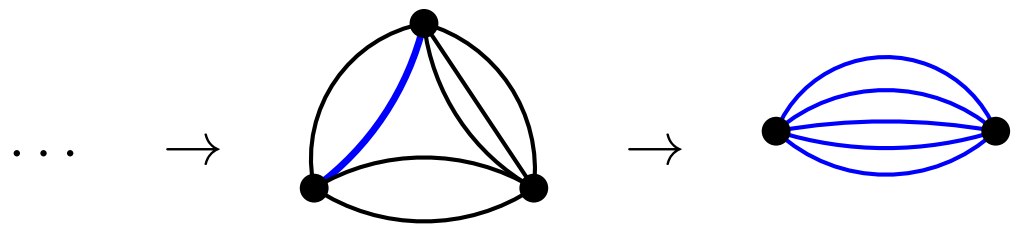
Mit $\lambda := 1$, haben wir

- **Zeit** $O(n^4)$
- **Erfolgsw'keit** $\geq 1 - e^{-1}$.

Kritische Phase im Cut-Algorithmus

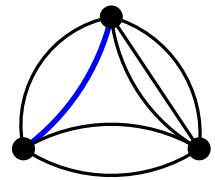
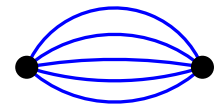
Cut(G) — G zusammenhängender Multigraph

1: $G' \leftarrow G$
2: **while** $|V(G')| > 2$ **do**
3: $\quad e \leftarrow$ gleichverteilt zufällige Kante in G'
4: $\quad G' \leftarrow G'/e$
5: **return** Grösse des eindeutigen Schnitts in G'



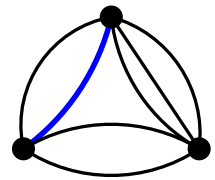
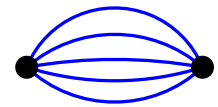
$$\hat{p}(n) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \underbrace{\frac{3}{5}}_{0.6} \cdot \underbrace{\frac{2}{4}}_{0.5} \cdot \underbrace{\frac{1}{3}}_{0.33} \cdot \underbrace{\hat{p}(2)}_{=1} = \frac{2}{n(n-1)}$$

Letzte Schritte sind am kritischten

$\cdots \quad \rightarrow$  $\rightarrow$ 

$$\hat{p}(n) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \underbrace{\frac{3}{5}}_{0.6} \cdot \underbrace{\frac{2}{4}}_{0.5} \cdot \underbrace{\frac{1}{3}}_{0.33} \cdot \underbrace{\hat{p}(2)}_{=1} = \frac{2}{n(n-1)}$$

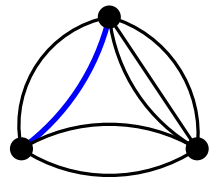
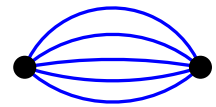
Letzte Schritte sind am kritischten

$\dots \quad \rightarrow$  $\rightarrow$ 

$$\hat{p}(n) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \underbrace{\frac{3}{5}}_{0.6} \cdot \underbrace{\frac{2}{4}}_{0.5} \cdot \underbrace{\frac{1}{3}}_{0.33} \cdot \underbrace{\hat{p}(2)}_{=1} = \frac{2}{n(n-1)}$$

Wir sollten Schritt von 3 auf 2 Knoten sorgfältiger machen

Letzte Schritte sind am kritischten

$\ldots \quad \rightarrow$  $\rightarrow$ 

$$\hat{p}(n) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \underbrace{\frac{3}{5}}_{0.6} \cdot \underbrace{\frac{2}{4}}_{0.5} \cdot \underbrace{\frac{1}{3}}_{0.33} \cdot \underbrace{\hat{p}(2)}_{=1} = \frac{2}{n(n-1)}$$

Wir sollten Schritt von 3 auf 2 Knoten sorgfältiger machen

. . . und am besten den Schritt von 4 auf 3 auch

. . .

Strategiewechsel im kritischen Bereich

Wir brechen bei G' mit t Knoten ab und verwenden den randomisierten $O(t^4)$ Algorithmus mit Erfolgsw'keit $\geq 1 - e^{-1}$.

$$\begin{aligned} \hat{p}_t(G) &:= \text{Wahrscheinlichkeit, dass Wert } \mu(G) \text{ ausgegeben wird} \\ \hat{p}_t(n) &:= \inf_{G=(V,E),\,|V|=n} \hat{p}_t(G), \qquad \hat{p}_t(t) \geq 1 - e^{-1} = \tfrac{e-1}{e} \end{aligned}$$

$$\hat{p}_t(n) \geq \tfrac{n-2}{n} \cdot \tfrac{n-3}{n-1} \cdot \tfrac{n-4}{n-2} \cdots \tfrac{t+1}{t+3} \cdot \tfrac{t}{t+2} \cdot \tfrac{t-1}{t+1} \cdot \hat{p}_t(t) \geq \underbrace{\frac{t(t-1)}{n(n-1)} \cdot \frac{e-1}{e}}_{\overline{p}_t(n) :=}$$

$\lambda \frac{1}{\overline{p}_t(n)}$-maliges Wiederholen gibt Fehlerw'keit $\leq e^{-\lambda}$ und Laufzeit

$$\overbrace{\lambda \frac{n(n-1)}{t(t-1)} \frac{e}{e-1}}^{\#\text{Wiederholungen}} \cdot O\Big(\underbrace{n(n-t)}_{\text{Reduktion auf } t \text{ Knoten}} + \overbrace{t^4}^{\text{Alg. auf } t \text{ Knoten}} \Big) = O\left(\lambda \left(\frac{n^4}{t^2} + n^2 t^2 \right) \right).$$

Optimierung von t

Erfolgsw'keit $\geq 1 - e^{-\lambda}$ und Laufzeit

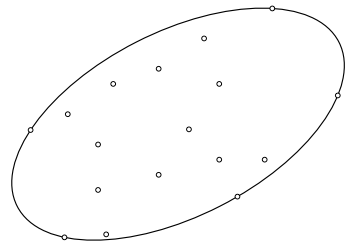
$$O(\lambda \underbrace{\left(\frac{n^4}{t^2} + n^2 t^2\right)}_{\to \min}) \overset{t=\sqrt{n}}{=} O(\lambda n^3)$$

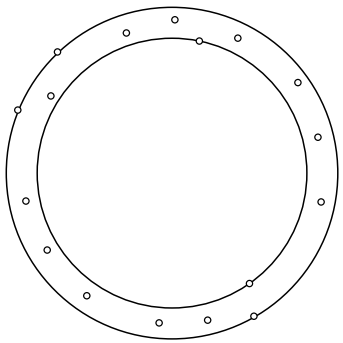
Bootstrapping: Es bietet sich an, die gleiche Methode nun mit dem neuen $O(n^3)$ Algorithmus statt dem $O(n^4)$ Algorithmus zu versuchen, und tatsächlich bekommen wir einen noch besseren, etc.

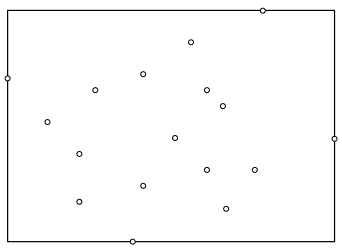
Im „Limit“ entwickelt sich so ein $O(n^2 \text{polylog}(n))$-Algorithmus.

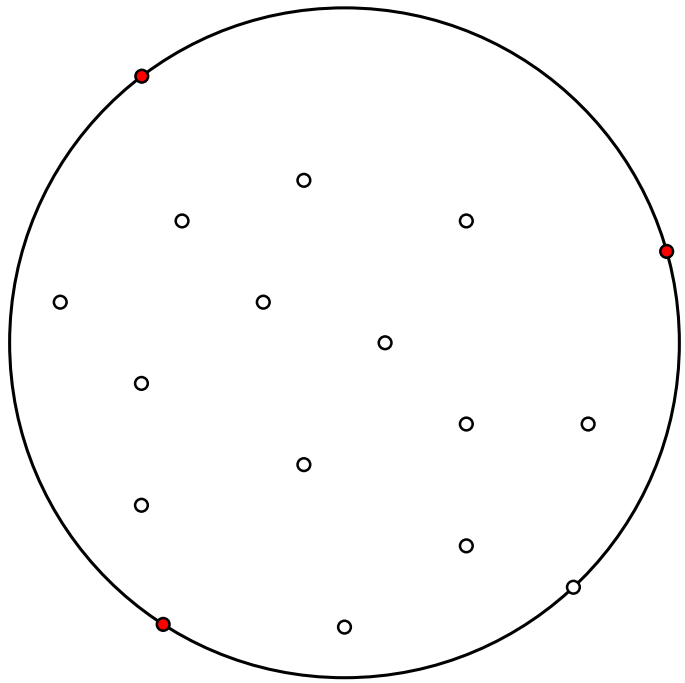
[Karger&Stein'96]

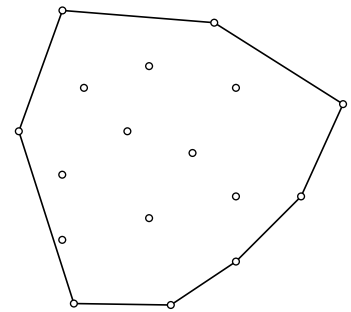
Kleinster umschliessender Kreis

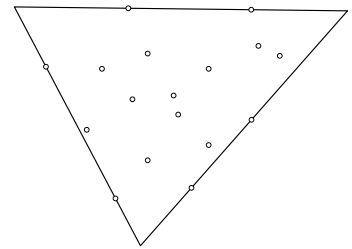


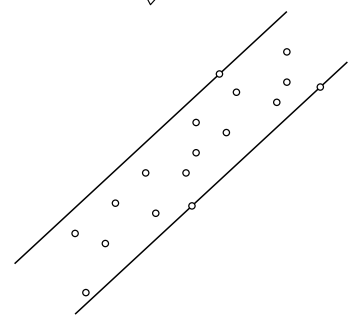












Problemstellung

SmallEnclDisk-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme den Kreis kleinsten Radius, der P umschliesst.

Problemstellung

SmallEnclDisk-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme den Kreis kleinsten Radius, der P umschliesst.

- Was heisst umschliessen?
- Gibt es so einen kleinsten Kreis überhaupt?
- Wenn ja, ist er eindeutig?

Problemstellung

SmallEnclDisk-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme den Kreis kleinsten Radius, der P umschliesst.

- Was heisst umschliessen?
- Gibt es so einen kleinsten Kreis überhaupt?
- Wenn ja, ist er eindeutig?

Für einen Kreis C, bezeichnen wir mit

$C^\bullet$ **die geschlossene von** C **berandete Kreisscheibe.**

Wir sagen C **umschliesst** P, falls $P \subseteq C^\bullet$. Punkte in P dürfen also auf C liegen.

Eindeutigkeit

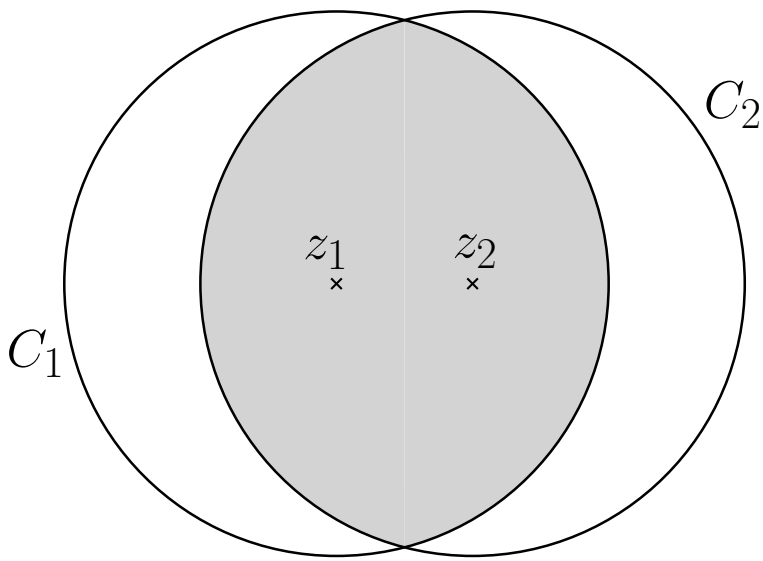
Lemma

Für jede endliche Punktemenge $P \subseteq \mathbb{R}^2$ gibt es einen eindeutigen **kleinsten umschliessenden Kreis** $C(P)$.

Eindeutigkeit

Lemma

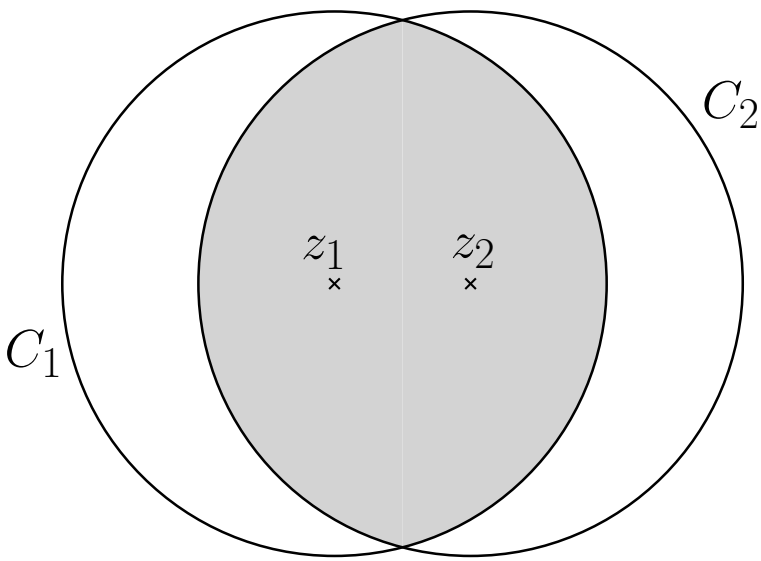
Für jede endliche Punktemenge $P \subseteq \mathbb{R}^2$ gibt es einen eindeutigen **kleinsten umschliessenden Kreis** $C(P)$.

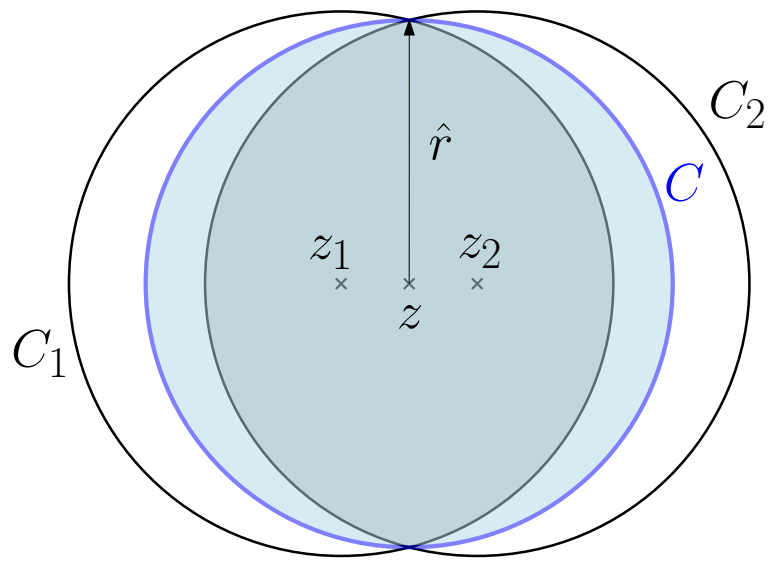


Eindeutigkeit

Lemma

Für jede endliche Punktemenge $P \subseteq \mathbb{R}^2$ gibt es einen eindeutigen **kleinsten umschliessenden Kreis** $C(P)$.





Eindeutigkeit

Lemma

Für jede endliche Punktemenge $P \subseteq \mathbb{R}^2$ gibt es einen eindeutigen **kleinsten umschliessenden Kreis** $C(P)$.

Beweis (nur für Eindeutigkeit). Angenommen, P hat zwei verschiedene kleinste umschliessende Kreise C_1 und C_2, beide mit Radius r und Mittelpunkten z_1 und z_2, $z_1 \neq z_2$. haben. Es gilt

$$P \subseteq C_1^\bullet \cap C_2^\bullet \ .$$

Sei C der Kreis mit Mittelpunkt $z = \frac{1}{2}(z_1 + z_2)$ (Mittelpunkt des Liniensegments zwischen z_1 und z_2). Sein Radius $\hat{r}$ sei der Abstand von z zu den beiden Schnittpunkten von C_1 und C_2. Dann gilt

$$P \subseteq C_1^\bullet \cap C_2^\bullet \subseteq C^\bullet \text{ und } \hat{r} = \sqrt{r^2 - \left(\frac{|z_1 z_2|}{2}\right)^2}$$

($|z_1 z_2|$ Abstand von z_1 zu z_2).

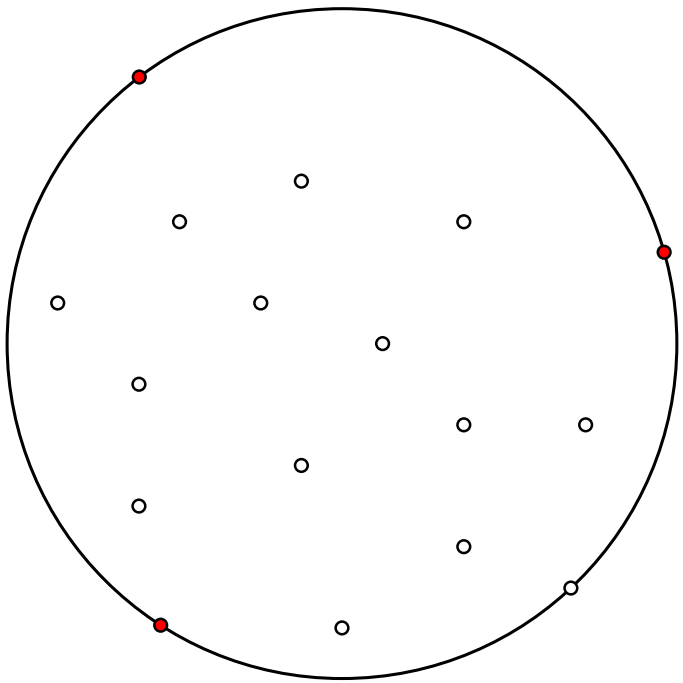
Da $\hat{r} < r$, sind C_1 und C_2 nicht kleinste umschl. Kreise. 

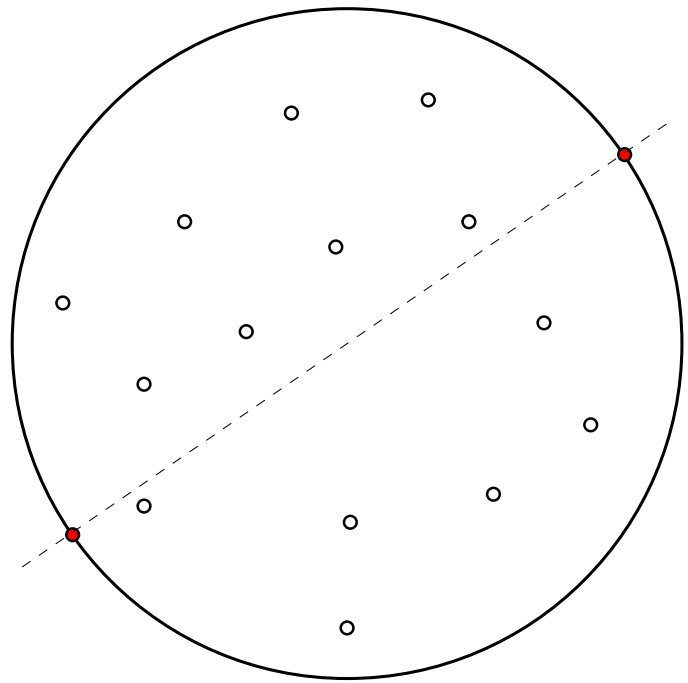
Kleine bestimmende Menge

Lemma

Für jede Punktemenge $P \subseteq \mathbb{R}^2$, $|P| \geq 3$, gibt es eine Teilmenge $Q \subseteq P$, so dass $|Q| = 3$ und $C(Q) = C(P)$. (Q Zertifikat für C(P).)

Hier ohne Beweis (siehe Skript).





Erste einfache Algorithmen

CompleteEnumeration(P)

1: **for all** $Q \in \binom{P}{3}$ **do**
2: bestimme $C(Q)$
3: **if** $P \subseteq C^\bullet(Q)$ **then**
4: **return** $C(Q)$

Erste einfache Algorithmen

CompleteEnumeration(P)

1: **for all** $Q \in \binom{P}{3}$ **do**
2: bestimme $C(Q)$
3: **if** $P \subseteq C^\bullet(Q)$ **then**
4: **return** $C(Q)$

Wir durchlaufen $\binom{n}{3}$ Mengen Q, berechnen $C(Q)$ in $O(1)$ Zeit, und prüfen $P \subseteq C^\bullet(Q)$ in $O(n)$ Zeit. Dies ergibt eine Laufzeit $O(n^4)$.

Erste einfache Algorithmen

CompleteEnumeration(P)

1: **for all** $Q \in \binom{P}{3}$ **do**
2: bestimme $C(Q)$
3: **if** $P \subseteq C^\bullet(Q)$ **then**
4: **return** $C(Q)$

Wir durchlaufen $\binom{n}{3}$ Mengen Q, berechnen $C(Q)$ in $O(1)$ Zeit, und prüfen $P \subseteq C^\bullet(Q)$ in $O(n)$ Zeit. Dies ergibt eine Laufzeit $O(n^4)$.

CompleteEnumerationSmart(P)

1: $r \leftarrow 0$
2: **for all** $Q \in \binom{P}{3}$ **do** bestimme $C(Q)$
3: **if** radius$(C(Q)) > r$ **then**
4: $C^* \leftarrow C(Q)$; $r \leftarrow$ radius$(C(Q))$
5: **return** C^*

Laufzeit $O(n^3)$.

Erster randomisierter Algorithmus

Randomised_PrimitiveVersion(P)

1: **repeat forever**
2: wähle $Q \subseteq P$ mit $|Q| = 3$ zufällig und gleichverteilt
3: bestimme $C(Q)$
4: **if** $P \subseteq C^{\bullet}(Q)$ **then**
5: **return** $C(Q)$

Erster randomisierter Algorithmus

Randomised_PrimitiveVersion(P)

1: **repeat forever**
2: wähle $Q \subseteq P$ mit $|Q| = 3$ zufällig und gleichverteilt
3: bestimme $C(Q)$
4: **if** $P \subseteq C^\bullet(Q)$ **then**
5: **return** $C(Q)$

Wir treffen auf das richtige Q mit Wahrscheinlichkeit $\geq 1/\binom{n}{3}$. Erwartete Anzahl der Versuche bis zum Erfolg ist $\leq \binom{n}{3}$, gibt erwartete Laufzeit $O(n^4)$.

Erster randomisierter Algorithmus

Randomised_PrimitiveVersion(P)

1: **repeat forever**
2: wähle $Q \subseteq P$ mit $|Q| = 3$ zufällig und gleichverteilt
3: bestimme $C(Q)$
4: **if** $P \subseteq C^{\bullet}(Q)$ **then**
5: **return** $C(Q)$

Wir treffen auf das richtige Q mit Wahrscheinlichkeit $\geq 1/\binom{n}{3}$. Erwartete Anzahl der Versuche bis zum Erfolg ist $\leq \binom{n}{3}$, gibt erwartete Laufzeit $O(n^4)$.

Idee: Wir lernen daraus, welche Punkte jeweils ausserhalb von $C(Q)$ liegen. Die sind intuitiv wichtiger für die Bestimmung von $C(P)$ als die Punkte innerhalb von $C(Q)$.

Der Algorithmus

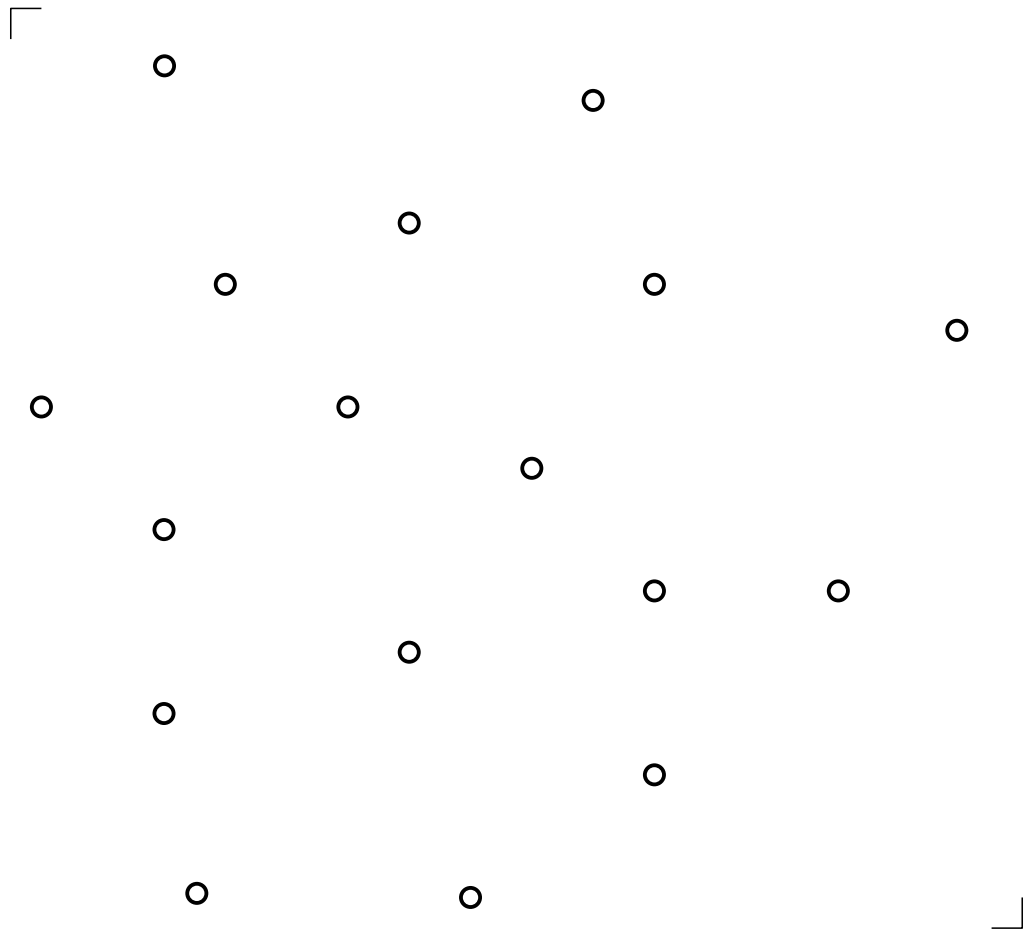
Randomised_CleverVersion(P)

1: $P' \leftarrow P$
2: **repeat**
3: wähle $Q \subseteq P'$ mit $|Q| = 11$ zufällig und gleichverteilt
4: bestimme $C(Q)$
5: **if** $P \subseteq C^\bullet(Q)$ **then return** $C(Q)$
6: **else** verdopple alle Punkte von P' ausserhalb von $C(Q)$
7: **forever**

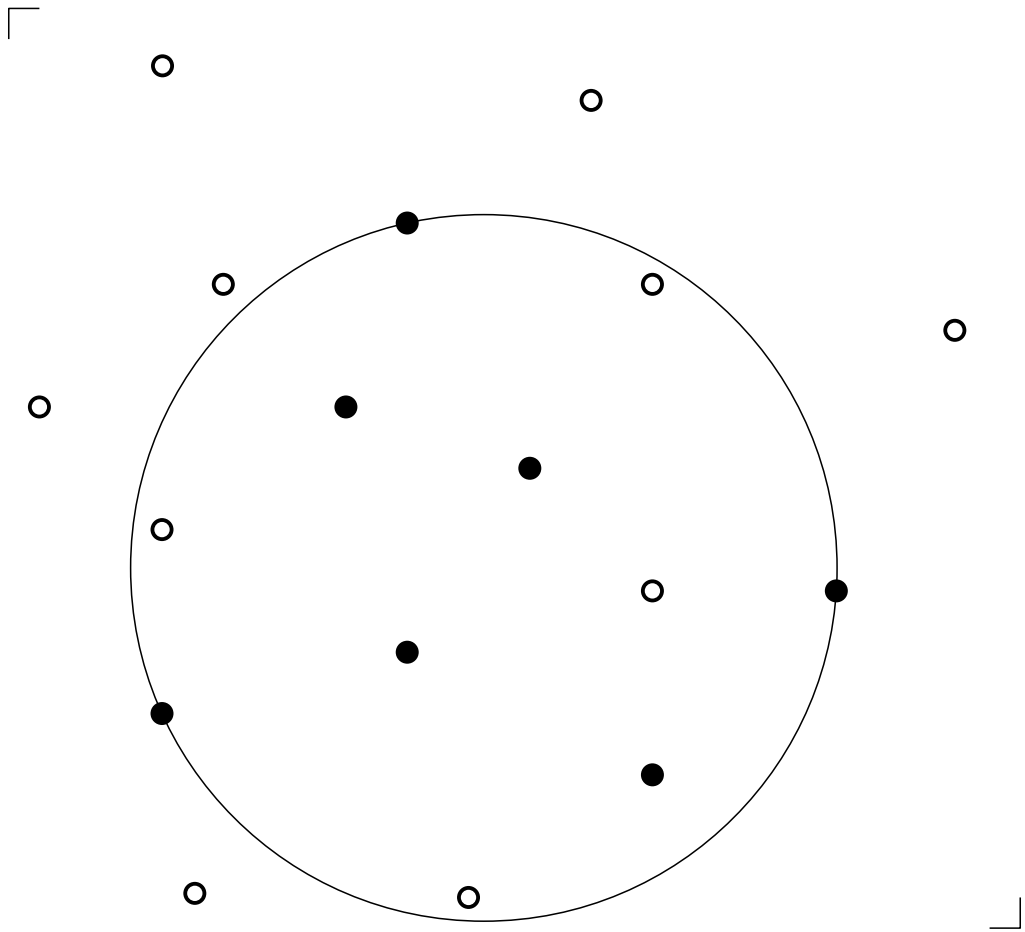
- Wir setzen voraus, dass die Wahl von Q in $O(n)$, $n := |P|$, möglich ist.
- So lässt sich jede Runde in $O(n)$ implementieren.

Wie viele Runden müssen wir machen, bis $P \subseteq C^\bullet(Q)$ gilt?

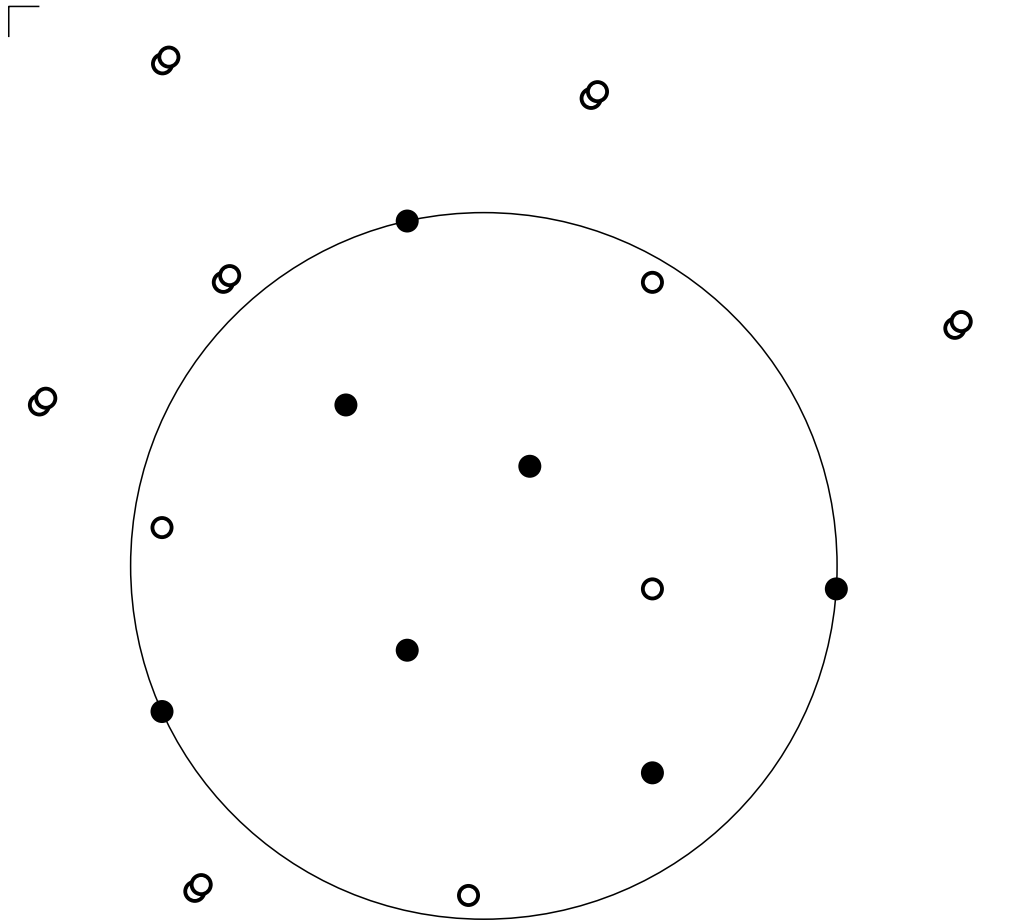
Algorithmus in Aktion



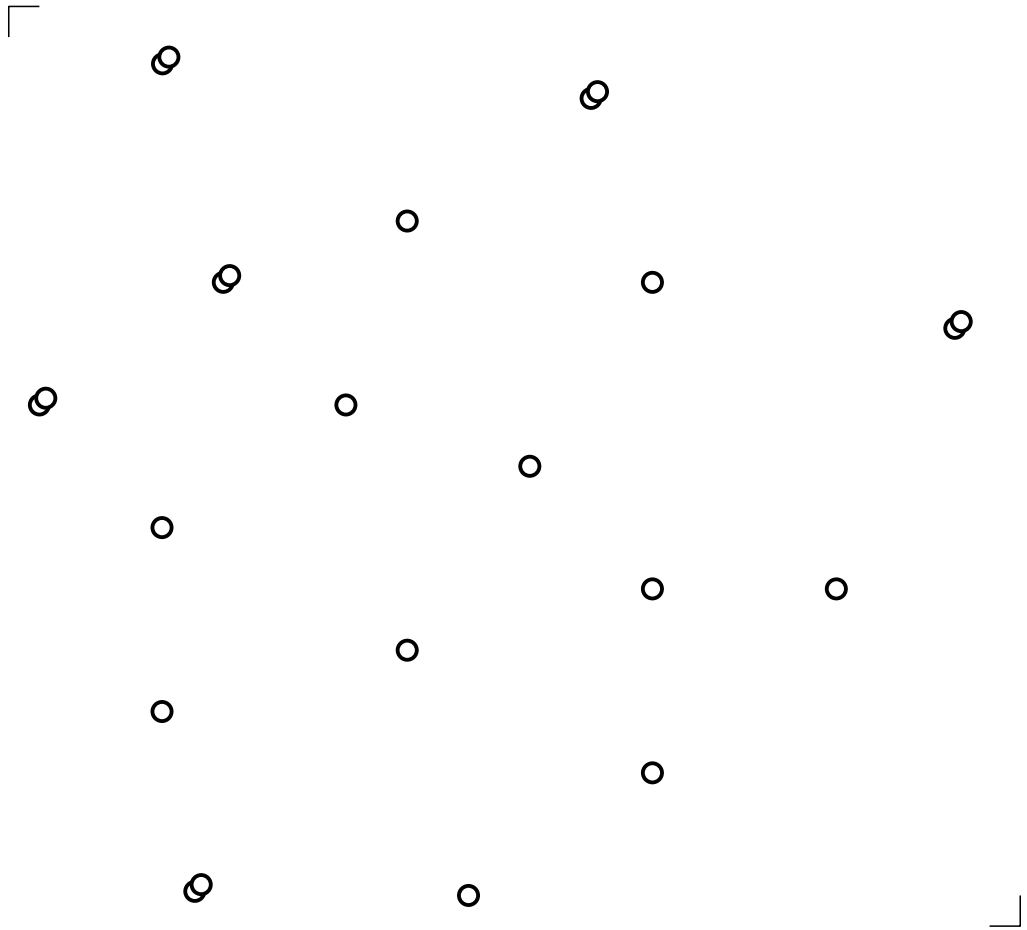
Algorithmus in Aktion



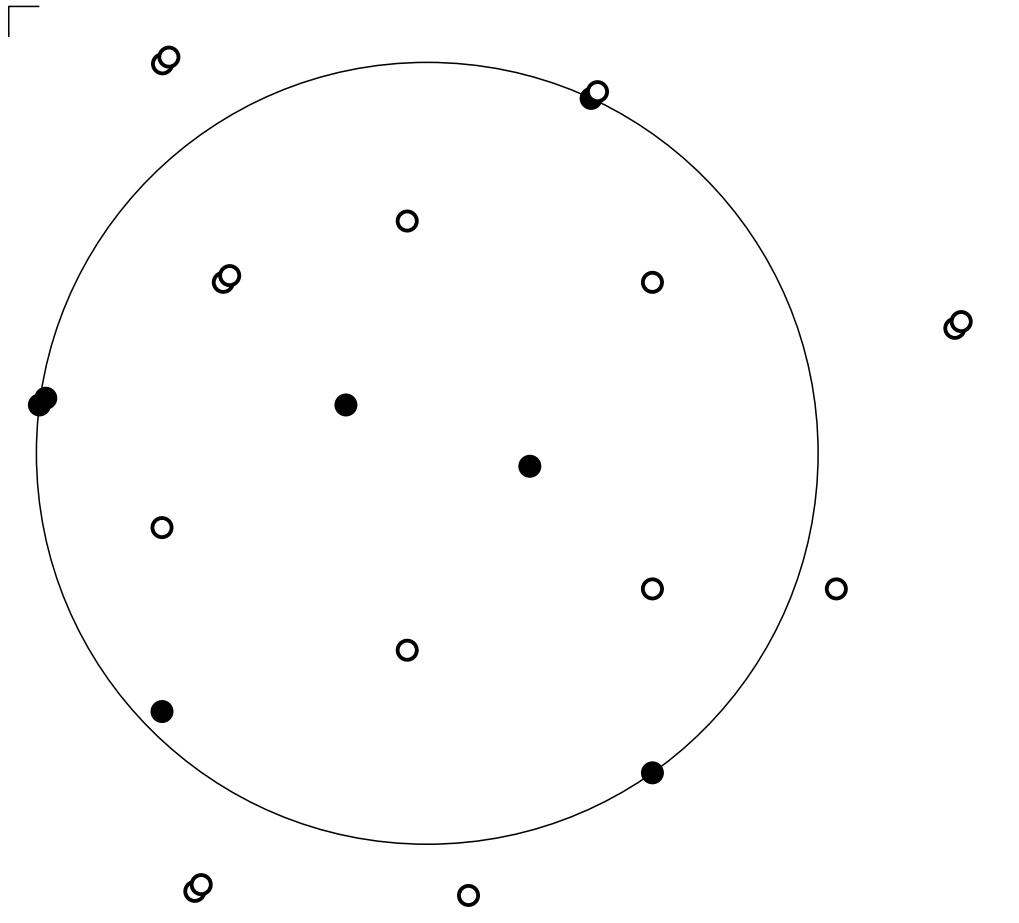
Algorithmus in Aktion



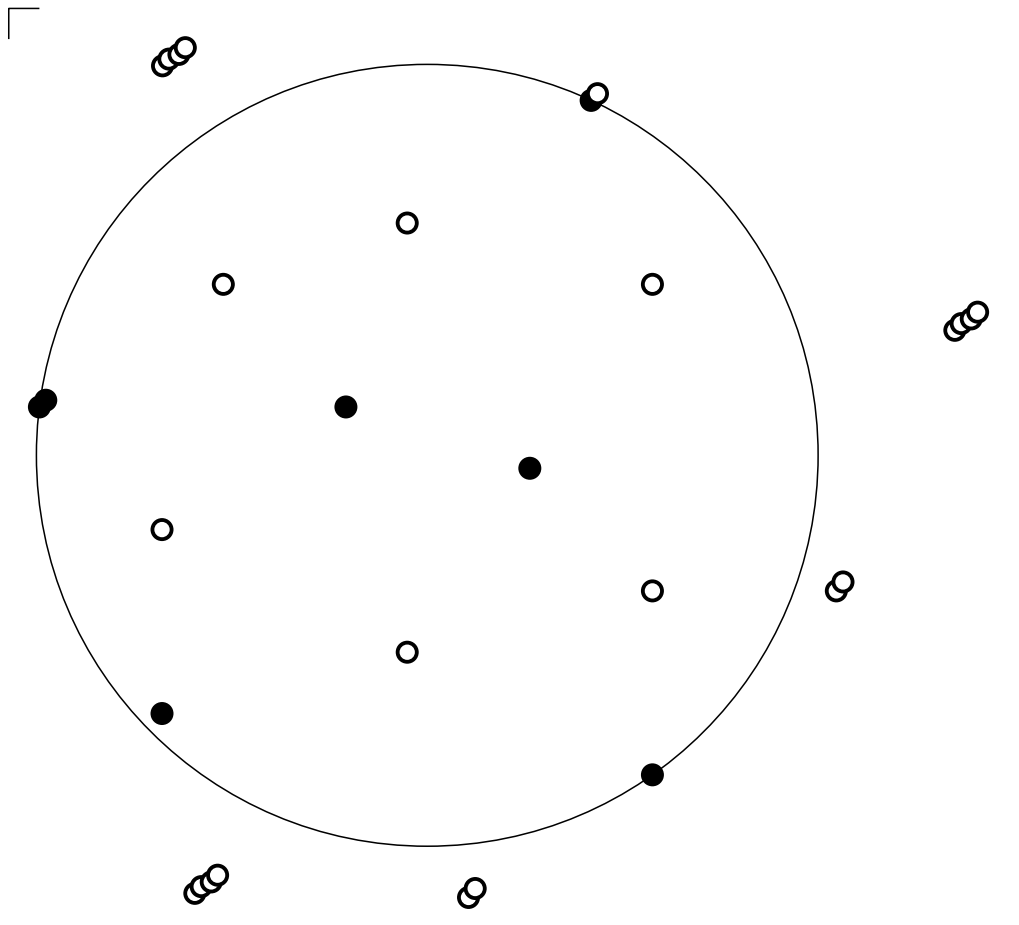
Algorithmus in Aktion



Algorithmus in Aktion



Algorithmus in Aktion



Zwei Kräfte spielen im Algorithmus gegeneinander

Sei $B^* \subseteq P$ mit $C(B^*) = C(P)$ und $|B^*| \leq 3$. Beachte:

$$B^* \subseteq Q \subseteq P \Rightarrow C(Q) = C(P).$$

Zwei Kräfte spielen im Algorithmus gegeneinander

Sei $B^* \subseteq P$ mit $C(B^*) = C(P)$ und $|B^*| \leq 3$. Beachte:

$$B^* \subseteq Q \subseteq P \Rightarrow C(Q) = C(P).$$

Wir betrachten P' nach k Runden:

- $|P'|$ **wächst stark.** Falls $P \subseteq C^\bullet(Q)$ nicht gilt, muss es einen Punkt aus B^* ausserhalb von $C(Q)$ geben. D.h. ein Punkt in B^* muss mindestens $k/3$ Mal verdoppelt worden sein und Vielfachkeit $\geq 2^{k/3}$ haben. Daher: $|P'| \geq \sqrt[3]{2}^k$.

Zwei Kräfte spielen im Algorithmus gegeneinander

Sei $B^* \subseteq P$ mit $C(B^*) = C(P)$ und $|B^*| \leq 3$. Beachte:

$$B^* \subseteq Q \subseteq P \Rightarrow C(Q) = C(P).$$

Wir betrachten P' nach k Runden:

- $|P'|$ **wächst stark.** Falls $P \subseteq C^\bullet(Q)$ nicht gilt, muss es einen Punkt aus B^* ausserhalb von $C(Q)$ geben. D.h. ein Punkt in B^* muss mindestens $k/3$ Mal verdoppelt worden sein und Vielfachkeit $\geq 2^{k/3}$ haben. Daher: $|P'| \geq \sqrt[3]{2}^k$.

- $|P'|$ **wächst nicht zu stark.** P' vergrössert sich in jeder Runde im Erwartungswert nur um einen Faktor $(1 + \frac{3}{12})$ (zu zeigen). Daher: $|P'| \leq \left(\frac{5}{4}\right)^k n$ (im einem gewissen probabilistischen Sinn).

Zwei Kräfte spielen im Algorithmus gegeneinander

Sei $B^* \subseteq P$ mit $C(B^*) = C(P)$ und $|B^*| \leq 3$. Beachte:

$$B^* \subseteq Q \subseteq P \Rightarrow C(Q) = C(P).$$

Wir betrachten P' nach k Runden:

- $|P'|$ **wächst stark.** Falls $P \subseteq C^\bullet(Q)$ nicht gilt, muss es einen Punkt aus B^* ausserhalb von $C(Q)$ geben. D.h. ein Punkt in B^* muss mindestens $k/3$ Mal verdoppelt worden sein und Vielfachkeit $\geq 2^{k/3}$ haben. Daher: $|P'| \geq \sqrt[3]{2}^k$.

- $|P'|$ **wächst nicht zu stark.** P' vergrössert sich in jeder Runde im Erwartungswert nur um einen Faktor $(1 + \frac{3}{12})$ (zu zeigen). Daher: $|P'| \leq \left(\frac{5}{4}\right)^k n$ (im einem gewissen probabilistischen Sinn).

Wegen $1.25 = \frac{5}{4} < \sqrt[3]{2} \approx 1.2599\ldots$ geht das nicht lange gut ...

Zwei Kräfte spielen im Algorithmus gegeneinander

Sei $B^* \subseteq P$ mit $C(B^*) = C(P)$ und $|B^*| \leq 3$. Beachte:

$$B^* \subseteq Q \subseteq P \Rightarrow C(Q) = C(P).$$

Wir betrachten P' nach k Runden:

- $|P'|$ **wächst stark.** Falls $P \subseteq C^\bullet(Q)$ nicht gilt, muss es einen Punkt aus B^* ausserhalb von $C(Q)$ geben. D.h. ein Punkt in B^* muss mindestens $k/3$ Mal verdoppelt worden sein und Vielfachkeit $\geq 2^{k/3}$ haben. Daher: $|P'| \geq \sqrt[3]{2}^k$.

- $|P'|$ **wächst nicht zu stark.** P' vergrössert sich in jeder Runde im Erwartungswert nur um einen Faktor $(1 + \frac{3}{12})$ (zu zeigen). Daher: $|P'| \leq \left(\frac{5}{4}\right)^k n$ (im einem gewissen probabilistischen Sinn).

Wegen $1.25 = \frac{5}{4} < \sqrt[3]{2} \approx 1.2599\ldots$ geht das nicht lange gut ...

und deshalb muss der Algorithmus „rechtzeitig“ terminieren.

Ein Sampling Lemma

Lemma

Sei $r, N \in \mathbb{N}$, $r \leq N$ und $P' \subseteq \mathbb{R}^2$ eine Multimenge, $|P'| = N$.
Für R zufällig gleichverteilt aus $\binom{P'}{r}$ gilt

$$\mathbb{E}(\,|\underbrace{P' \setminus C^\bullet(R)}|\,) \leq 3\frac{N-r}{r+1} \leq 3\frac{N}{r+1}$$

Punkte in P' ausserhalb von $C(R)$

Ein Sampling Lemma

Lemma

Sei $r, N \in \mathbb{N}$, $r \leq N$ und $P' \subseteq \mathbb{R}^2$ eine Multimenge, $|P'| = N$.
Für R zufällig gleichverteilt aus $\binom{P'}{r}$ gilt

$$\mathbb{E}(|\underbrace{P' \setminus C^\bullet(R)}_{\text{Punkte in } P' \text{ ausserhalb von } C(R)}|) \leq 3\frac{N-r}{r+1} \leq 3\frac{N}{r+1}$$

Beweis. Für $p \in P'$, $R, Q \subseteq P'$ definiere Indikatorvariablen

$$out(p,R) := \begin{cases} 1\,, p \notin C^\bullet(R) \\ 0\,, \text{sonst.} \end{cases} \quad ess(p,Q) := \begin{cases} 1\,, C(Q \setminus \{p\}) \neq C(Q) \\ 0\,, \text{sonst.} \end{cases}$$

Ein Sampling Lemma

Lemma

Sei $r, N \in \mathbb{N}$, $r \leq N$ und $P' \subseteq \mathbb{R}^2$ eine Multimenge, $|P'| = N$.
Für R zufällig gleichverteilt aus $\binom{P'}{r}$ gilt

$$\mathbb{E}(\big|\underbrace{P' \setminus C^\bullet(R)}_{\text{Punkte in } P' \text{ ausserhalb von } C(R)}\big|) \leq 3\frac{N-r}{r+1} \leq 3\frac{N}{r+1}$$

Beweis. Für $p \in P'$, $R, Q \subseteq P'$ definiere Indikatorvariablen

$$out(p,R) := \begin{cases} 1\,, p \notin C^\bullet(R) \\ 0\,, \text{sonst.} \end{cases} \quad ess(p,Q) := \begin{cases} 1\,, C(Q \setminus \{p\}) \neq C(Q) \\ 0\,, \text{sonst.} \end{cases}$$

- $\sum_{p \in P' \setminus R} out(p,R) = \big|P' \setminus C^\bullet(R)\big|$.

Ein Sampling Lemma

Lemma

Sei $r, N \in \mathbb{N}$, $r \leq N$ und $P' \subseteq \mathbb{R}^2$ eine Multimenge, $|P'| = N$.
Für R zufällig gleichverteilt aus $\binom{P'}{r}$ gilt

$$\mathbb{E}(\big|\underbrace{P' \setminus C^\bullet(R)}_{\text{Punkte in } P' \text{ ausserhalb von } C(R)}\big|) \leq 3\frac{N-r}{r+1} \leq 3\frac{N}{r+1}$$

Beweis. Für $p \in P'$, $R, Q \subseteq P'$ definiere Indikatorvariablen

$$out(p,R) := \begin{cases} 1\,, p \notin C^\bullet(R) \\ 0\,, \text{sonst.} \end{cases} \quad ess(p,Q) := \begin{cases} 1\,, C(Q \setminus \{p\}) \neq C(Q) \\ 0\,, \text{sonst.} \end{cases}$$

- $\sum_{p \in P' \setminus R} out(p,R) = \big|P' \setminus C^\bullet(R)\big|$.
- $\sum_{p \in Q} ess(p,Q) \leq 3$.

Ein Sampling Lemma

Lemma

Sei $r, N \in \mathbb{N}$, $r \leq N$ und $P' \subseteq \mathbb{R}^2$ eine Multimenge, $|P'| = N$.
Für R zufällig gleichverteilt aus $\binom{P'}{r}$ gilt

$$\mathbb{E}(\big|\underbrace{P' \setminus C^\bullet(R)}_{\text{Punkte in } P' \text{ ausserhalb von } C(R)}\big|) \leq 3\frac{N-r}{r+1} \leq 3\frac{N}{r+1}$$

Beweis. Für $p \in P'$, $R, Q \subseteq P'$ definiere Indikatorvariablen

$$out(p,R) := \begin{cases} 1\,, p \notin C^\bullet(R) \\ 0\,, \text{sonst.} \end{cases} \quad ess(p,Q) := \begin{cases} 1\,, C(Q \setminus \{p\}) \neq C(Q) \\ 0\,, \text{sonst.} \end{cases}$$

- $\sum_{p \in P' \setminus R} out(p,R) = \big|P' \setminus C^\bullet(R)\big|$.
- $\sum_{p \in Q} ess(p,Q) \leq 3$.
- $out(p,R) = 1 \iff ess(p, R \cup \{p\}) = 1$.

Fortsetzung des Beweises des Sampling Lemmas

- $\sum_{p \in P' \setminus R} out(p, R) = \left|P' \setminus C^\bullet(R)\right|$.
- $\sum_{p \in Q} ess(p, Q) \leq 3$.
- $out(p, R) = 1 \iff ess(p, R \cup \{p\}) = 1$.

$$\mathbb{E}[\underbrace{\left|P' \setminus C^\bullet(R)\right|}_{\text{Zufallsvariable}}] = \frac{1}{\binom{N}{r}} \sum_{R \in \binom{P'}{r}} \sum_{s \in P' \setminus R} out(s, R)$$

Fortsetzung des Beweises des Sampling Lemmas

- $\sum_{p \in P' \setminus R} out(p, R) = \left| P' \setminus C^\bullet(R) \right|$.
- $\sum_{p \in Q} ess(p, Q) \leq 3$.
- $out(p, R) = 1 \iff ess(p, R \cup \{p\}) = 1$.

$$\mathbb{E}[\underbrace{\left| P' \setminus C^\bullet(R) \right|}_{\text{Zufallsvariable}}] = \frac{1}{\binom{N}{r}} \sum_{R \in \binom{P'}{r}} \sum_{s \in P' \setminus R} out(s, R)$$

$$= \frac{1}{\binom{N}{r}} \sum_{R \in \binom{P'}{r}} \sum_{s \in P' \setminus R} ess(s, R \cup \{s\})$$

Fortsetzung des Beweises des Sampling Lemmas

- $\sum_{p \in P' \setminus R} out(p, R) = \left| P' \setminus C^{\bullet}(R) \right|$.
- $\sum_{p \in Q} ess(p, Q) \leq 3$.
- $out(p, R) = 1 \iff ess(p, R \cup \{p\}) = 1$.

$$
\begin{aligned}
\mathbb{E}[\underbrace{\left| P' \setminus C^{\bullet}(R) \right|}_{\text{Zufallsvariable}}] &= \frac{1}{\binom{N}{r}} \sum_{R \in \binom{P'}{r}} \sum_{s \in P' \setminus R} out(s, R) \\
&= \frac{1}{\binom{N}{r}} \sum_{R \in \binom{P'}{r}} \sum_{s \in P' \setminus R} ess(s, R \cup \{s\}) \\
&= \frac{1}{\binom{N}{r}} \sum_{Q \in \binom{P'}{r+1}} \underbrace{\sum_{p \in Q} ess(p, Q)}_{\leq 3}
\end{aligned}
$$

Fortsetzung des Beweises des Sampling Lemmas

- $\sum_{p \in P' \setminus R} out(p, R) = \left| P' \setminus C^\bullet(R) \right|$.
- $\sum_{p \in Q} ess(p, Q) \leq 3$.
- $out(p, R) = 1 \iff ess(p, R \cup \{p\}) = 1$.

$$
\begin{aligned}
\mathbb{E}[\underbrace{\left| P' \setminus C^\bullet(R) \right|}_{\text{Zufallsvariable}}] &= \frac{1}{\binom{N}{r}} \sum_{R \in \binom{P'}{r}} \sum_{s \in P' \setminus R} out(s, R) \\
&= \frac{1}{\binom{N}{r}} \sum_{R \in \binom{P'}{r}} \sum_{s \in P' \setminus R} ess(s, R \cup \{s\}) \\
&= \frac{1}{\binom{N}{r}} \sum_{Q \in \binom{P'}{r+1}} \underbrace{\sum_{p \in Q} ess(p, Q)}_{\leq 3} \\
&\leq \frac{1}{\binom{N}{r}} \sum_{Q \in \binom{P'}{r+1}} 3 = 3 \cdot \frac{\binom{N}{r+1}}{\binom{N}{r}} = 3 \frac{N - r}{r + 1}
\end{aligned}
$$

Anzahl Punkte nach k Runden

Sei $T \in \mathbb{N} \cup \{\infty\}$ die Anzahl der Runden des Algorithmus, und $X_k := |P'|$, für P' nach $\min\{T, k\}$ Runden ($X_0 \equiv |P| = n$). Es gilt

(wir schreiben r für 11)

$$\mathbb{E}[X_k] \quad = \quad \sum_{t=0}^{\infty} \mathbb{E}[X_k \mid X_{k-1} = t] \cdot \Pr[X_{k-1} = t]$$

Anzahl Punkte nach k Runden

Sei $T \in \mathbb{N} \cup \{\infty\}$ die Anzahl der Runden des Algorithmus, und $X_k := |P'|$, für P' nach $\min\{T, k\}$ Runden ($X_0 \equiv |P| = n$). Es gilt

(wir schreiben r für 11)

$$\begin{aligned}
\mathbb{E}[X_k] &= \sum_{t=0}^{\infty} \mathbb{E}[X_k \mid X_{k-1} = t] \cdot \Pr[X_{k-1} = t] \\
&\leq \sum_{t=0}^{\infty} \left(1 + \frac{3}{r+1}\right) t \cdot \Pr[X_{k-1} = t]
\end{aligned}$$

Anzahl Punkte nach k Runden

Sei $T \in \mathbb{N} \cup \{\infty\}$ die Anzahl der Runden des Algorithmus, und $X_k := |P'|$, für P' nach $\min\{T, k\}$ Runden ($X_0 \equiv |P| = n$). Es gilt

(wir schreiben r für 11)

$$\begin{aligned}
\mathbb{E}[X_k] &= \sum_{t=0}^{\infty} \mathbb{E}[X_k \mid X_{k-1} = t] \cdot \Pr[X_{k-1} = t] \\
&\leq \sum_{t=0}^{\infty} \left(1 + \frac{3}{r+1}\right) t \cdot \Pr[X_{k-1} = t] \\
&= \left(1 + \frac{3}{r+1}\right) \cdot \sum_{t=0}^{\infty} t \cdot \Pr[X_{k-1} = t]
\end{aligned}$$

Anzahl Punkte nach k Runden

Sei $T \in \mathbb{N} \cup \{\infty\}$ die Anzahl der Runden des Algorithmus, und $X_k := |P'|$, für P' nach $\min\{T, k\}$ Runden ($X_0 \equiv |P| = n$). Es gilt

(wir schreiben r für 11)

$$\begin{aligned}
\mathbb{E}[X_k] &= \sum_{t=0}^{\infty} \mathbb{E}[X_k \mid X_{k-1} = t] \cdot \Pr[X_{k-1} = t] \\
&\leq \sum_{t=0}^{\infty} \left(1 + \frac{3}{r+1}\right) t \cdot \Pr[X_{k-1} = t] \\
&= \left(1 + \frac{3}{r+1}\right) \cdot \sum_{t=0}^{\infty} t \cdot \Pr[X_{k-1} = t] \\
&= \left(1 + \frac{3}{r+1}\right) \cdot \mathbb{E}[X_{k-1}]
\end{aligned}$$

Anzahl Punkte nach k Runden

Sei $T \in \mathbb{N} \cup \{\infty\}$ die Anzahl der Runden des Algorithmus, und $X_k := |P'|$, für P' nach $\min\{T, k\}$ Runden ($X_0 \equiv |P| = n$). Es gilt

(wir schreiben r für 11)

$$\begin{aligned}
\mathbb{E}[X_k] &= \sum_{t=0}^{\infty} \mathbb{E}[X_k \mid X_{k-1} = t] \cdot \Pr[X_{k-1} = t] \\
&\leq \sum_{t=0}^{\infty} \left(1 + \frac{3}{r+1}\right) t \cdot \Pr[X_{k-1} = t] \\
&= \left(1 + \frac{3}{r+1}\right) \cdot \sum_{t=0}^{\infty} t \cdot \Pr[X_{k-1} = t] \\
&= \left(1 + \frac{3}{r+1}\right) \cdot \mathbb{E}[X_{k-1}]
\end{aligned}$$

Wegen $X_0 \equiv n$ folgt $\mathbb{E}[X_k] \leq (1 + \frac{3}{r+1})^k \cdot n$.

Zusammenfassung

Wir erinnern uns: Falls $T \geq k$, dann $X_k \geq \sqrt[3]{2}^k$.

Zusammenfassung

Wir erinnern uns: Falls $T \geq k$, dann $X_k \geq \sqrt[3]{2}^k$.

- Einerseits $\mathbb{E}[X_k] \leq (1 + \frac{3}{11+1})^k \cdot n$.

Zusammenfassung

Wir erinnern uns: Falls $T \geq k$, dann $X_k \geq \sqrt[3]{2}^k$.

- Einerseits $\mathbb{E}[X_k] \leq (1 + \frac{3}{11+1})^k \cdot n$.
- Andererseits

$$\begin{aligned}\mathbb{E}[X_k] &= \mathbb{E}[\underbrace{X_k \mid T \geq k}_{\geq 2^{k/3}}] \cdot \Pr[T \geq k] + \underbrace{\mathbb{E}[X_k \mid T < k]}_{\geq 0} \cdot \Pr[T < k] \\ &\geq \quad 2^{k/3} \cdot \Pr[T \geq k] \ .\end{aligned}$$

Zusammenfassung

Wir erinnern uns: Falls $T \geq k$, dann $X_k \geq \sqrt[3]{2}^k$.

- Einerseits $\mathbb{E}[X_k] \leq (1 + \frac{3}{11+1})^k \cdot n$.
- Andererseits

$$\begin{aligned}\mathbb{E}[X_k] &= \mathbb{E}[\underbrace{X_k \mid T \geq k}_{\geq 2^{k/3}}] \cdot \Pr[T \geq k] + \underbrace{\mathbb{E}[X_k \mid T < k]}_{\geq 0} \cdot \Pr[T < k] \\ &\geq \quad 2^{k/3} \cdot \Pr[T \geq k] \ .\end{aligned}$$

- Also $\sqrt[3]{2}^k \cdot \Pr[T \geq k] \leq \mathbb{E}[X_k] \leq \left(\frac{5}{4}\right)^k \cdot n$.

Zusammenfassung

Wir erinnern uns: Falls $T \geq k$, dann $X_k \geq \sqrt[3]{2}^k$.

- Einerseits $\mathbb{E}[X_k] \leq (1 + \frac{3}{11+1})^k \cdot n$.
- Andererseits

$$\begin{aligned}\mathbb{E}[X_k] &= \mathbb{E}[\underbrace{X_k \mid T \geq k}_{\geq 2^{k/3}}] \cdot \Pr[T \geq k] + \underbrace{\mathbb{E}[X_k \mid T < k]}_{\geq 0} \cdot \Pr[T < k] \\ &\geq \quad 2^{k/3} \cdot \Pr[T \geq k] \ .\end{aligned}$$

- Also $\sqrt[3]{2}^k \cdot \Pr[T \geq k] \leq \mathbb{E}[X_k] \leq \left(\frac{5}{4}\right)^k \cdot n$.

$$\Pr[T \geq k] \leq \Big(\underbrace{\frac{5}{4\sqrt[3]{2}}}_{\leq 0.993} \Big)^k \cdot n \leq 0.993^k n \qquad \underbrace{\leq 1}_{\text{falls } k \geq -\log_{0.993} n}$$

Finale

Satz

Der Algorithmus berechnet den kleinsten umschliessenden Kreis in erwartet $O(n \log n)$ Zeit.

Beweis. Die Laufzeit ist $O(nT)$. Setze $k_0 := \lfloor -\log_{0.993} n \rfloor$.

$$\mathbb{E}[T] = \sum_{k \geq 1} \Pr[T \geq k]$$

Finale

Satz

Der Algorithmus berechnet den kleinsten umschliessenden Kreis in erwartet $O(n \log n)$ Zeit.

Beweis. Die Laufzeit ist $O(nT)$. Setze $k_0 := \lfloor -\log_{0.993} n \rfloor$.

$$\begin{aligned} \mathbb{E}[T] &= \sum_{k \geq 1} \Pr[T \geq k] \\ &\leq \sum_{k=1}^{k_0} 1 + \sum_{k > k_0} 0.993^k n \end{aligned}$$

Finale

Satz

Der Algorithmus berechnet den kleinsten umschliessenden Kreis in erwartet $O(n \log n)$ Zeit.

Beweis. Die Laufzeit ist $O(nT)$. Setze $k_0 := \lfloor -\log_{0.993} n \rfloor$.

$$
\begin{aligned}
\mathbb{E}[T] &= \sum_{k \geq 1} \Pr[T \geq k] \\
&\leq \sum_{k=1}^{k_0} 1 + \sum_{k > k_0} 0.993^k n \\
&= \underbrace{\sum_{k=1}^{k_0} 1}_{=k_0} + \sum_{k' \geq 0} 0.993^{k'} \cdot \underbrace{0.993^{k_0+1} n}_{\leq 1}
\end{aligned}
$$

Finale

Satz

Der Algorithmus berechnet den kleinsten umschliessenden Kreis in erwartet $O(n \log n)$ Zeit.

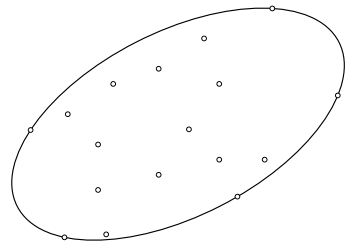
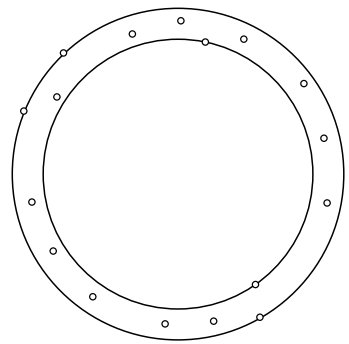
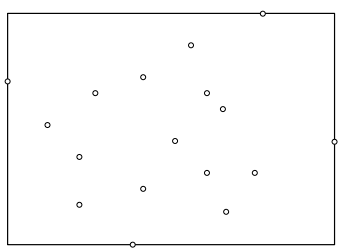
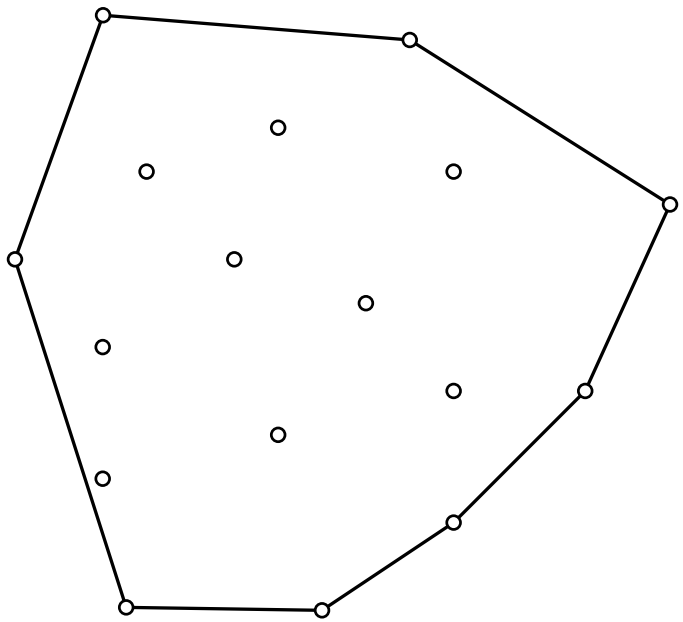
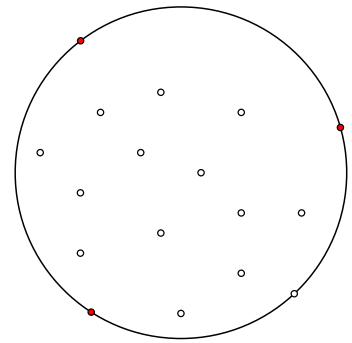
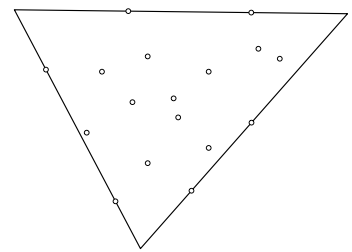
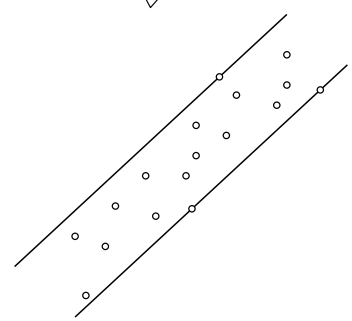
Beweis. Die Laufzeit ist $O(nT)$. Setze $k_0 := \lfloor -\log_{0.993} n \rfloor$.

$$
\begin{aligned}
\mathbb{E}[T] &= \sum_{k \geq 1} \Pr[T \geq k] \\
&\leq \sum_{k=1}^{k_0} 1 + \sum_{k > k_0} 0.993^k n \\
&= \underbrace{\sum_{k=1}^{k_0} 1}_{=k_0} + \sum_{k' \geq 0} 0.993^{k'} \cdot \underbrace{0.993^{k_0+1} n}_{\leq 1} \\
&= k_0 + O(1) = O(\log n)
\end{aligned}
$$

Anmerkungen

- Der Algorithmus nach einer Idee von [Clarkson'95] funktioniert für die kleinste umschliessende Kugel (oder Ellipse) in jeder Dimension, mit jeweils entsprechenden anderen Konstanten statt „11“. Bei fixer Dimension bleibt die Laufzeit $O(n \log n)$.
- Es gibt andere einfache randomisierte und auch deterministische Linearzeitalgorithmen (in jeder fixer Dimension).

Konvexe Hülle: Jarvis Wrap

Problemstellung

ConvexHull-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme die konvexe Hülle von P.

Problemstellung

ConvexHull-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme die konvexe Hülle von P.

- Was ist die konvexe Hülle?
- Wie stellt man die konvexe Hülle dar?

Problemstellung

ConvexHull-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme die konvexe Hülle von P.

- Was ist die konvexe Hülle?
- Wie stellt man die konvexe Hülle dar?

- Wie geht man mit Spezialfällen und numerischen Problemen um?

Konvexe Menge, konvexe Hülle

Sei $d \in \mathbb{N}$.

- Für $v_0, v_1 \in \mathbb{R}^d$ sei

$$\overline{v_0 v_1} := \{(1-\lambda)v_0 + \lambda v_1 \mid \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1\} ,$$

  das v_0 und v_1 verbindende **Liniensegment**.

Konvexe Menge, konvexe Hülle

Sei $d \in \mathbb{N}$.

- Für $v_0, v_1 \in \mathbb{R}^d$ sei

$$\overline{v_0 v_1} := \{(1-\lambda)v_0 + \lambda v_1 \mid \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1\} ,$$

  das v_0 und v_1 verbindende **Liniensegment**.

- Eine Menge $C \subseteq \mathbb{R}^d$ heisst **konvex**, falls

$$\forall v_0, v_1 \in C : \ \overline{v_0 v_1} \subseteq C .$$

Konvexe Menge, konvexe Hülle

Sei $d \in \mathbb{N}$.

- Für $v_0, v_1 \in \mathbb{R}^d$ sei

$$\overline{v_0 v_1} := \{(1-\lambda)v_0 + \lambda v_1 \mid \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1\} ,$$

  das v_0 und v_1 verbindende **Liniensegment**.

- Eine Menge $C \subseteq \mathbb{R}^d$ heisst **konvex**, falls

$$\forall v_0, v_1 \in C: \ \overline{v_0 v_1} \subseteq C .$$

- Die **konvexe Hülle**, $\mathrm{conv}(S)$, einer Menge $S \subseteq \mathbb{R}^d$ ist der Schnitt aller konvexen Mengen, die S enthalten, d.h.

$$\mathrm{conv}(S) := \bigcap_{S \subseteq C \subseteq \mathbb{R}^d,\ C \text{ konvex}} C .$$

Konvexe Menge, konvexe Hülle

Sei $d \in \mathbb{N}$.

- Für $v_0, v_1 \in \mathbb{R}^d$ sei

$$\overline{v_0 v_1} := \{(1-\lambda)v_0 + \lambda v_1 \mid \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1\} ,$$

  das v_0 und v_1 verbindende **Liniensegment**.

- Eine Menge $C \subseteq \mathbb{R}^d$ heisst **konvex**, falls

$$\forall v_0, v_1 \in C : \ \overline{v_0 v_1} \subseteq C .$$

- Die **konvexe Hülle**, $\mathrm{conv}(S)$, einer Menge $S \subseteq \mathbb{R}^d$ ist der Schnitt aller konvexen Mengen, die S enthalten, d.h.

$$\mathrm{conv}(S) := \bigcap_{S \subseteq C \subseteq \mathbb{R}^d,\ C \text{ konvex}} C .$$

siehe „Diskrete Mathematik“, „Analysis“, „Lineare Algebra“.

Darstellung der konvexen Hülle

Für eine endliche Punktemenge P in der Ebene wird die konvexe Hülle durch ein Polygon, der Rand von $\text{conv}(P)$, bestimmt, dessen Ecken Punkte aus P sind. Wenn wir von der Berechnung von $\text{conv}(P)$ sprechen, so meinen wir die Bestimmung einer Folge

$$(q_0, q_1, \ldots, q_{h-1}), \quad h \leq n,$$

der Ecken dieses Polygons, beginnend bei einer beliebiger Ecke q_0 und dann entgegen dem Uhrzeigersinn entlang dieses Polygons.

Darstellung der konvexen Hülle

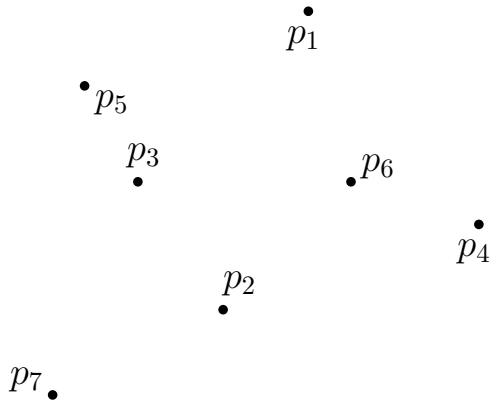
Für eine endliche Punktemenge P in der Ebene wird die konvexe Hülle durch ein Polygon, der Rand von $\mathrm{conv}(P)$, bestimmt, dessen Ecken Punkte aus P sind. Wenn wir von der Berechnung von $\mathrm{conv}(P)$ sprechen, so meinen wir die Bestimmung einer Folge

$$(q_0, q_1, \ldots, q_{h-1}), \quad h \leq n,$$

der Ecken dieses Polygons, beginnend bei einer beliebiger Ecke q_0 und dann entgegen dem Uhrzeigersinn entlang dieses Polygons.

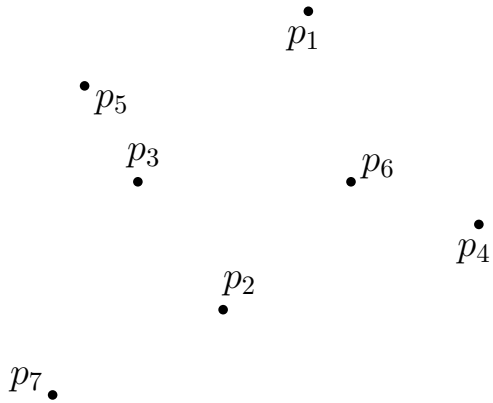
Beachte: $Q := \{q_0, q_1, \ldots, q_{h-1}\} \subseteq P$ ist die kleinste Teilmenge von P mit $\mathrm{conv}(Q) = \mathrm{conv}(P)$.

Problemstellung

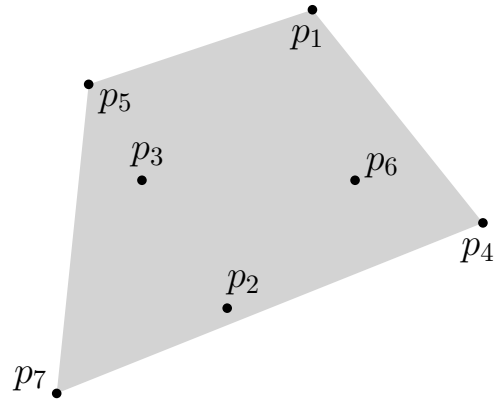


Punktemenge P,

Problemstellung

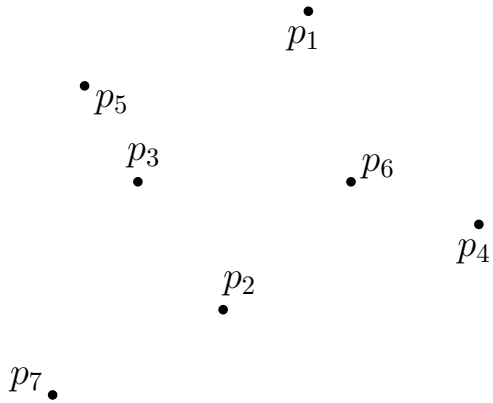


Punktemenge P,

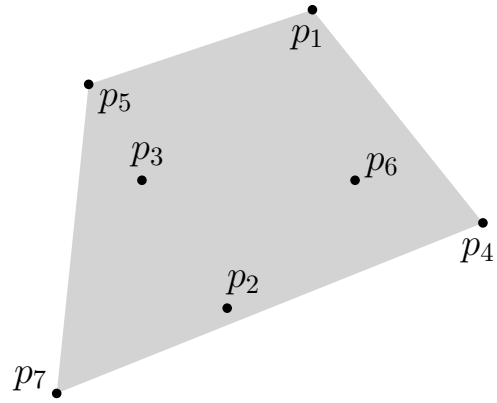


konvexe Hülle $\text{conv}(P)$,

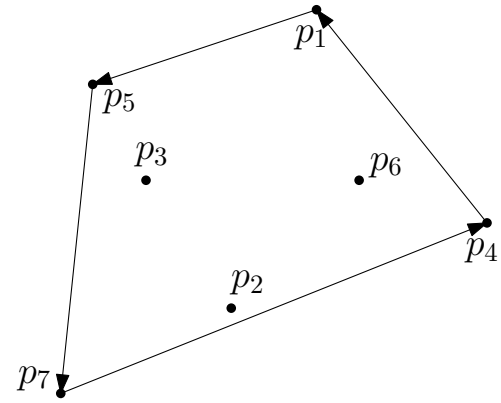
Problemstellung



Punktemenge P,

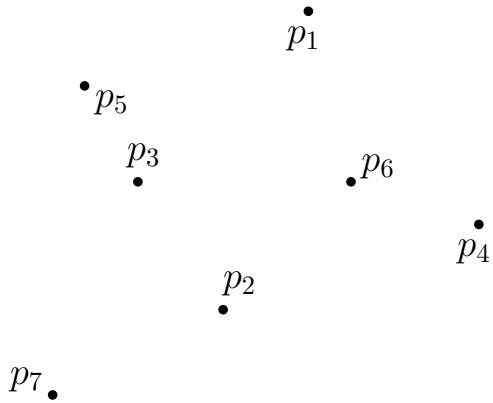


konvexe Hülle $\text{conv}(P)$,

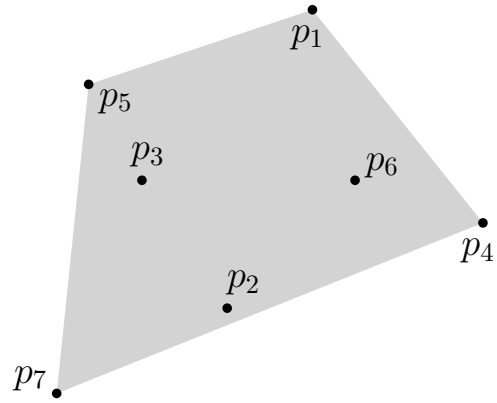


Polygon (p_4, p_1, p_5, p_7).

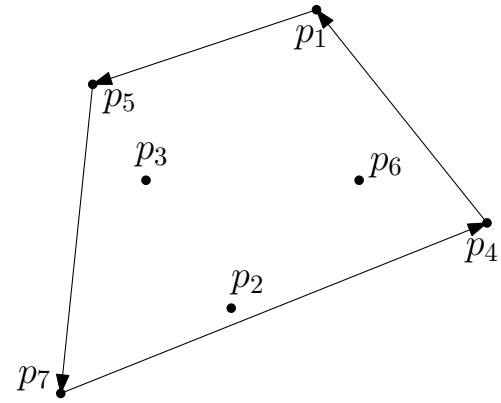
Problemstellung



Punktemenge P,



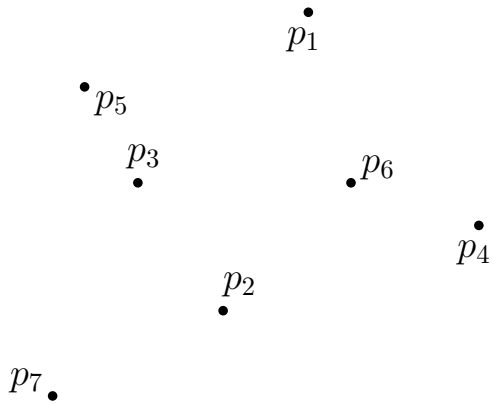
konvexe Hülle $\text{conv}(P)$,



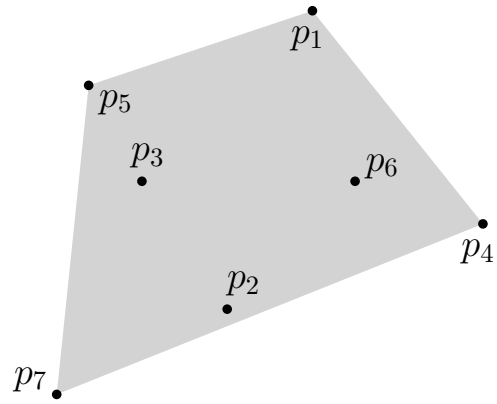
Polygon (p_4, p_1, p_5, p_7).

ConvexHull-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme die Ecken des $\text{conv}(P)$ umrandenden Polygons, in der Reihenfolge gegen den Uhrzeigersinn.

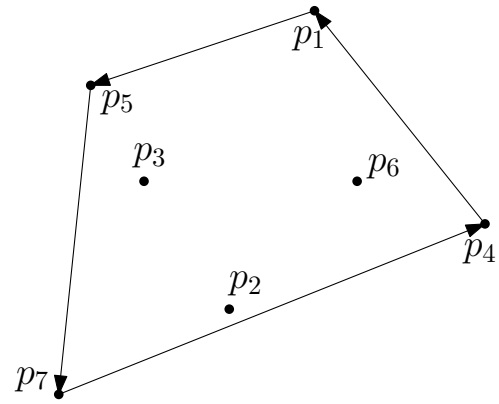
Problemstellung



Punktemenge P,



konvexe Hülle $\text{conv}(P)$,

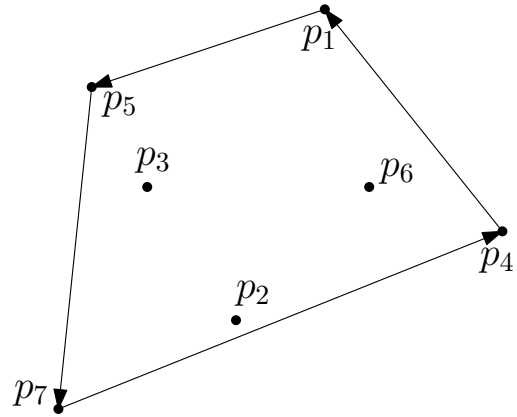


Polygon (p_4, p_1, p_5, p_7).

ConvexHull-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme die Ecken des $\text{conv}(P)$ umrandenden Polygons, in der Reihenfolge gegen den Uhrzeigersinn.

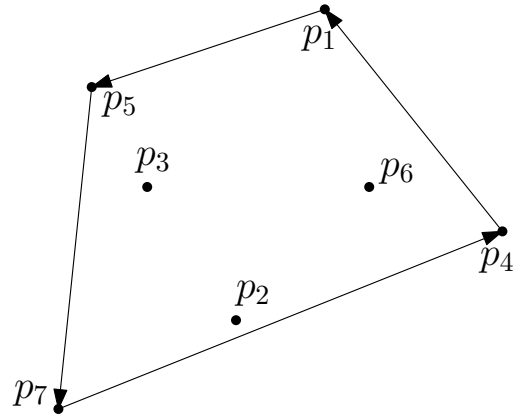
Vereinfachende Annahme: **Allgemeine Lage**, d.h. keine 3 Punkte auf einer gemeinsamen Geraden, keine 2 Pkt gleiche x-Koordinate.

Randkanten



Ein Paar $qr \in P^2$, $q \neq r$, heisst **Randkante** von P, falls alle Punkte in $P \setminus \{q, r\}$ links von qr liegen, d.h. auf der linken Seite der gerichteten Geraden durch q und r, gerichtet von q nach r, liegen.

Randkanten



Ein Paar $qr \in P^2$, $q \neq r$, heisst **Randkante** von P, falls alle Punkte in $P \setminus \{q, r\}$ links von qr liegen, d.h. auf der linken Seite der gerichteten Geraden durch q und r, gerichtet von q nach r, liegen.

Lemma

$(q_0, q_1, \ldots, q_{h-1})$ *ist die Eckenfolge des* $\mathrm{conv}(P)$ *umschliessenden Polygons gegen den Uhrzeigersinn genau dann wenn alle Paare* (q_{i-1}, q_i), $i = 1, 2, \ldots, h$, *Randkanten von* P *sind (Indizes* $\bmod h$*).*

Orientierungstest

Wie entscheiden wir „p liegt links von qr“?

Orientierungstest

Wie entscheiden wir „p liegt links von qr“?

Lemma

Seien $p = (p_x, p_y)$, $q = (q_x, q_y)$, und $r = (r_x, r_y)$ Punkte in $\mathbb{R}^2$. Es gilt $q \neq r$ und p liegt links von qr genau dann wenn

$$\det(p,q,r) := \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} = \begin{vmatrix} q_x - p_x & q_y - p_y \\ r_x - p_x & r_y - p_y \end{vmatrix} > 0$$

$$\Leftrightarrow \quad (q_x - p_x)(r_y - p_y) > (q_y - p_y)(r_x - p_x)$$

Orientierungstest

Wie entscheiden wir „p liegt links von qr“?

Lemma

Seien $p = (p_x, p_y)$, $q = (q_x, q_y)$, und $r = (r_x, r_y)$ Punkte in $\mathbb{R}^2$. Es gilt $q \neq r$ und p liegt links von qr genau dann wenn

$$\det(p,q,r) := \begin{vmatrix} p_x & p_y & 1 \\ q_x & q_y & 1 \\ r_x & r_y & 1 \end{vmatrix} = \begin{vmatrix} q_x - p_x & q_y - p_y \\ r_x - p_x & r_y - p_y \end{vmatrix} > 0$$

$$\Leftrightarrow \quad (q_x - p_x)(r_y - p_y) > (q_y - p_y)(r_x - p_x)$$

$\frac{1}{2}|\det(p,q,r)| =$ die Fläche des Dreiecks pqr

Das Vorzeichen von $\det(p,q,r)$ bestimmt, wie die Punkte p, q, r den Rand dieses Dreiecks durchlaufen.

Erster naiver Ansatz

Gehe durch jedes der $n(n-1)$ geordneten Paare qr, und prüfe, ob dies eine Randkante ist,

Erster naiver Ansatz

Gehe durch jedes der $n(n-1)$ geordneten Paare qr, und prüfe, ob dies eine Randkante ist,

> indem man für alle $n-2$ Punkte p in $P \setminus \{q, r\}$ feststellt, ob p links von qr liegt.

So haben wir die Randkanten in $O(n^3)$ gefunden, die wir nur mehr richtig aneinanderreihen müssen.

Finden des nächsten Punkts

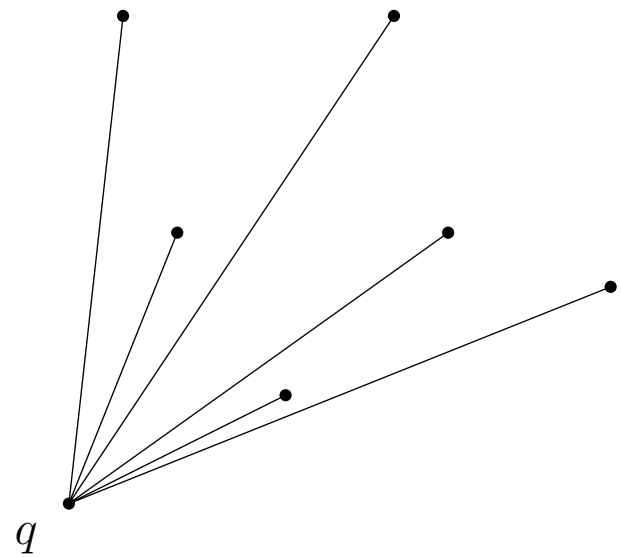
$q_0 :=$ Punkt mit kleinster x-Koordinate in P.

q_0 ist sicher eine Ecke der konvexen Hülle, also Teil der gesuchten Folge und wir können insbesondere die Folge auch mit q_0 beginnen.

Wie finden wir den Punkt q_1, der die Randkante q_0q_1 bildet?

FindNext(q)

1: Wähle $p_0 \in P \setminus \{q\}$ beliebig
2: $q_{\text{next}} \leftarrow p_0$
3: **for all** $p \in P \setminus \{q, p_0\}$ **do**
4: **if** p rechts von qq_{next} **then**
5: $q_{\text{next}} \leftarrow p$
6: **return** q_{next}



Ordnung um einen Punkt

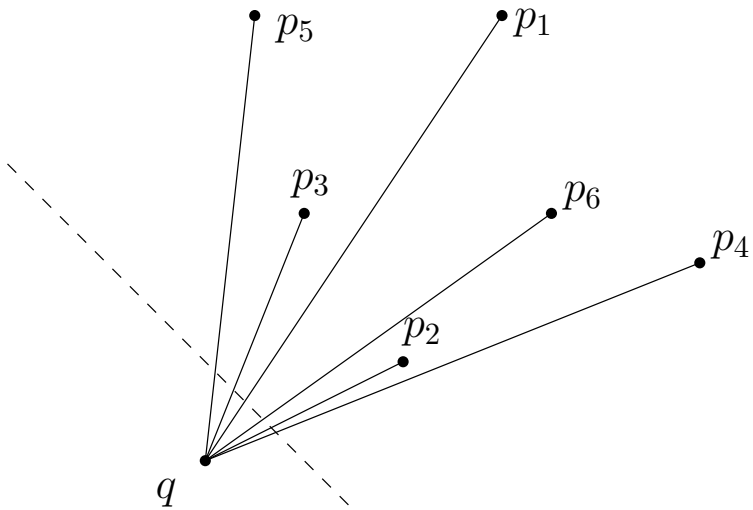
Gegeben $q \in P$, sei $\prec_q$ eine Relation auf $P \setminus \{q\}$ mittels

$$p_1 \prec_q p_2 \quad :\Leftrightarrow \quad p_1 \text{ rechts von } qp_2$$

Ordnung um einen Punkt

Gegeben $q \in P$, sei $\prec_q$ eine Relation auf $P \setminus \{q\}$ mittels

$$p_1 \prec_q p_2 \quad :\Leftrightarrow \quad p_1 \text{ rechts von } qp_2$$

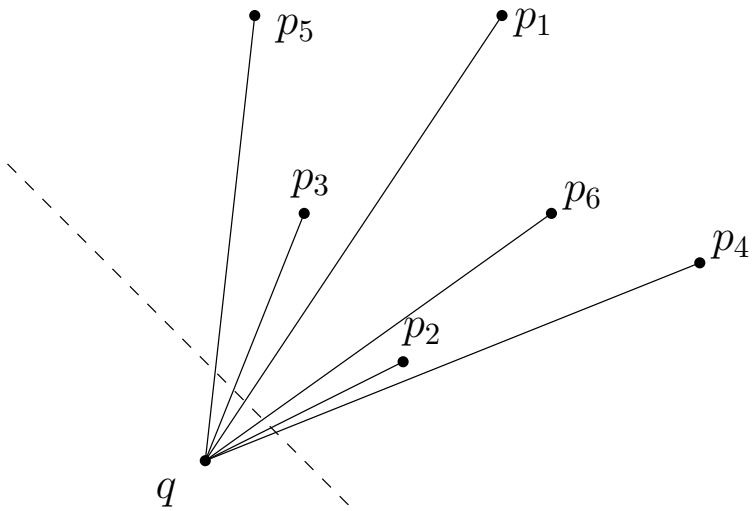


$$p_4 \prec_q p_2 \prec_q p_6 \prec_q p_1 \prec_q p_3 \prec_q p_5$$

Ordnung um einen Punkt

Gegeben $q \in P$, sei $\prec_q$ eine Relation auf $P \setminus \{q\}$ mittels

$$p_1 \prec_q p_2 \quad :\Leftrightarrow \quad p_1 \text{ rechts von } qp_2$$



$$p_4 \prec_q p_2 \prec_q p_6 \prec_q p_1 \prec_q p_3 \prec_q p_5$$

qp_4 ist Randkante.

Lemma

Ist q eine Ecke der konvexen Hülle von P, so ist die Relation $\prec_q$ eine totale Ordnung auf $P \setminus \{q\}$. Für das Minimum $p_{\min}$ dieser Ordnung gilt, dass $qp_{\min}$ eine Randkante ist.

Jarvis Wrap (Einwickeln)

JarvisWrap(P)

1: $h \leftarrow 0$
2: $p_{\text{now}} \leftarrow$ Punkt in P mit kleinster x-Koordinate
3: **repeat**
4: $\quad q_h \leftarrow p_{\text{now}}$
5: $\quad p_{\text{now}} \leftarrow \text{FindNext}(q_h)$
6: $\quad h \leftarrow h + 1$
7: **until** $p_{\text{now}} = q_0$
8: **return** $(q_0, q_1, \ldots, q_{h-1})$

Jarvis Wrap (Einwickeln)

JarvisWrap(P)

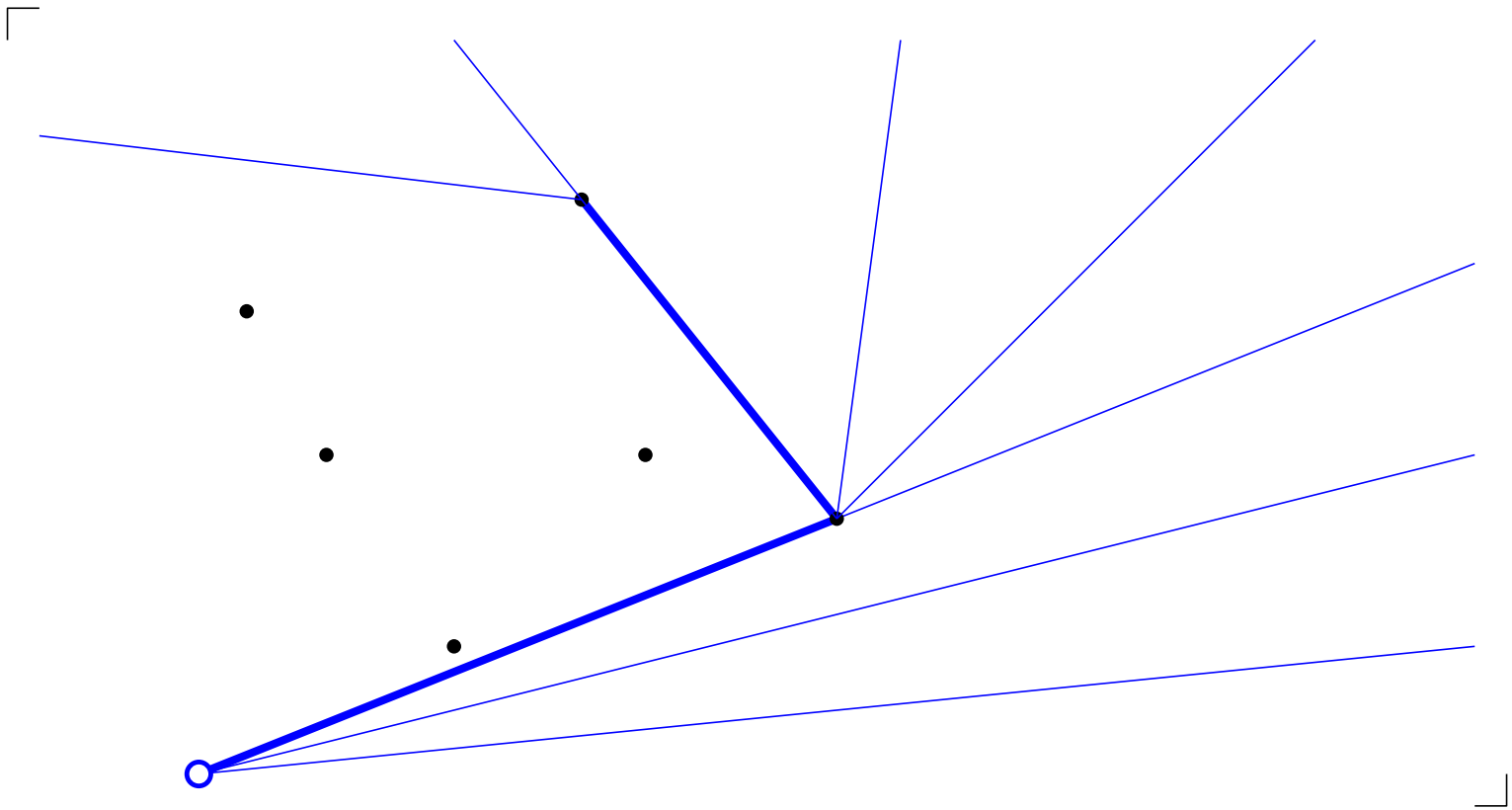
1: $h \leftarrow 0$
2: $p_{\text{now}} \leftarrow$ Punkt in P mit kleinster x-Koordinate
3: **repeat**
4: $\quad q_h \leftarrow p_{\text{now}}$
5: $\quad p_{\text{now}} \leftarrow \text{FindNext}(q_h)$
6: $\quad h \leftarrow h + 1$
7: **until** $p_{\text{now}} = q_0$
8: **return** $(q_0, q_1, \ldots, q_{h-1})$

Satz

Für eine Menge P von n Punkten in allgemeiner Lage in $\mathbb{R}^2$ berechnet der Algorithmus JarvisWrap *die konvexe Hülle in Zeit $O(nh)$, wobei h die Anzahl der Ecken der konvexen Hülle von P ist.*

Einwickeln (wrap)

Einwickeln (wrap)



Jarvis Wrap – Laufzeit

Satz

Für eine Menge P von n Punkten in allgemeiner Lage in $\mathbb{R}^2$ berechnet der Algorithmus JarvisWrap *die konvexe Hülle in Zeit $O(nh)$, wobei h die Anzahl der Ecken der konvexen Hülle von P ist.*

Jarvis Wrap – Laufzeit

Satz

Für eine Menge P von n Punkten in allgemeiner Lage in $\mathbb{R}^2$ berechnet der Algorithmus JarvisWrap *die konvexe Hülle in Zeit $O(nh)$, wobei h die Anzahl der Ecken der konvexen Hülle von P ist.*

- Da $h \leq n$, läuft JarvisWrap in $O(n^2)$ (statt $O(n^3)$).
- Ist $h = O(1)$, z.B. $\text{conv}(P)$ ist ein Dreieck, läuft der Algorithmus in $O(n)$ Zeit.

Jarvis Wrap – Laufzeit

Satz

Für eine Menge P von n Punkten in allgemeiner Lage in $\mathbb{R}^2$ berechnet der Algorithmus JarvisWrap *die konvexe Hülle in Zeit $O(nh)$, wobei h die Anzahl der Ecken der konvexen Hülle von P ist.*

- Da $h \leq n$, läuft JarvisWrap in $O(n^2)$ (statt $O(n^3)$).
- Ist $h = O(1)$, z.B. $\text{conv}(P)$ ist ein Dreieck, läuft der Algorithmus in $O(n)$ Zeit.

Für Punkte zufällig in einem Quadrat: Erwartete #Ecken $O(\log n)$.

Jarvis Wrap – Laufzeit

Satz

Für eine Menge P von n Punkten in allgemeiner Lage in $\mathbb{R}^2$ berechnet der Algorithmus JarvisWrap *die konvexe Hülle in Zeit $O(nh)$, wobei h die Anzahl der Ecken der konvexen Hülle von P ist.*

- Da $h \leq n$, läuft JarvisWrap in $O(n^2)$ (statt $O(n^3)$).
- Ist $h = O(1)$, z.B. $\text{conv}(P)$ ist ein Dreieck, läuft der Algorithmus in $O(n)$ Zeit.

Für Punkte zufällig in einem Quadrat: Erwartete #Ecken $O(\log n)$.

Für Punkte zufällig in einer Kreisscheibe: Erwartete #Ecken $O(\sqrt[3]{n})$.

Implementierung – Degeneriertheiten

Kollinearitäten (3 Punkte auf Gerade), **gleiche x-Koord.**, ...

- Anfangspunkt q_0 als den Punkt mit lexikographisch kleinster Koordinate (unter allen mit kleinster x-Koordinate, den mit kleinster y-Koordinate). (Adaption der x-Ordnung der Punkte)
- Der Test "p rechts von qq_{next}" muss ersetzt werden durch (p rechts von qq_{next}) oder (p auf der Geraden durch qq_{next} und $|qp| > |qq_{\text{next}}|$). (Adaption der Ordnung $\prec_q$)
- In der Regel können wir nicht einmal annehmen, dass die Punkte verschieden sind (z.B. gegeben in einem Feld)!

Software ohne Berücksichtigung dieser Fälle hat geringen Nutzen!

Implementierung – numerische Probleme

$$\text{“}(q_x - p_x)(r_y - p_y) > (q_y - p_y)(r_x - p_x)\text{”}$$

ist in einer Implementierung mit Fliesskommazahlen nicht exakt.

Implementierung – numerische Probleme

$$\text{“}(q_x - p_x)(r_y - p_y) > (q_y - p_y)(r_x - p_x)\text{”}$$

ist in einer Implementierung mit Fliesskommazahlen nicht exakt.

Es geht oft nicht um die absolute Genauigkeit des Ergebnisses (z.B. bei Eingaben, die selbst schon mit Fehlern behaftet sind). Vielmehr kann der Algorithmus völlig falsche Ergebnisse liefern oder in eine unendliche Schleife laufen.

Implementierung – numerische Probleme

$$\text{“}(q_x - p_x)(r_y - p_y) > (q_y - p_y)(r_x - p_x)\text{”}$$

ist in einer Implementierung mit Fliesskommazahlen nicht exakt.

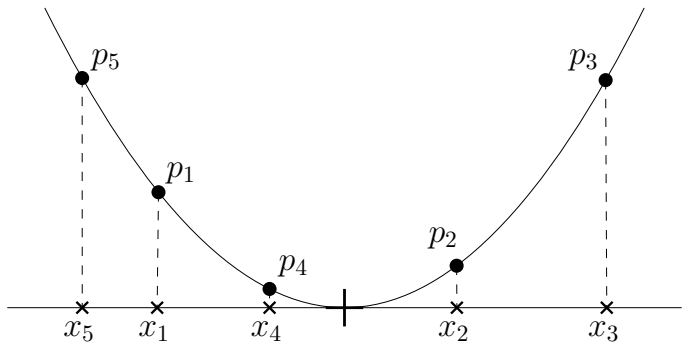
Es geht oft nicht um die absolute Genauigkeit des Ergebnisses (z.B. bei Eingaben, die selbst schon mit Fehlern behaftet sind). Vielmehr kann der Algorithmus völlig falsche Ergebnisse liefern oder in eine unendliche Schleife laufen.

Zum Beispiel **Vorbeilaufen am Startpunkt** (wegen numerischer Probleme, weil Startpunkt doppelt auftritt).

Programmbibliotheken bieten exakte Datentypen
(für spezielle Operationen).

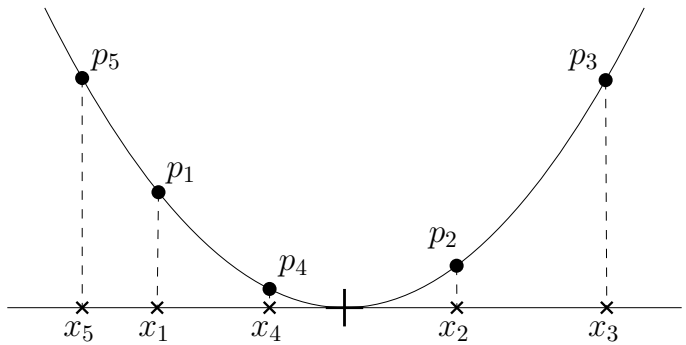
Untere Schranke für ConvexHull

Betrachte eine Folge $(x_1, x_2, \ldots, x_n)$ von Zahlen in $\mathbb{R}$. Wir setzen $p_i = (x_i, x_i^2)$, $i = 1, 2, \ldots, n$ (vertikale Projektion von der x-Achse im $\mathbb{R}^2$ auf die Einheitsparabel $y = x^2$).



Untere Schranke für ConvexHull

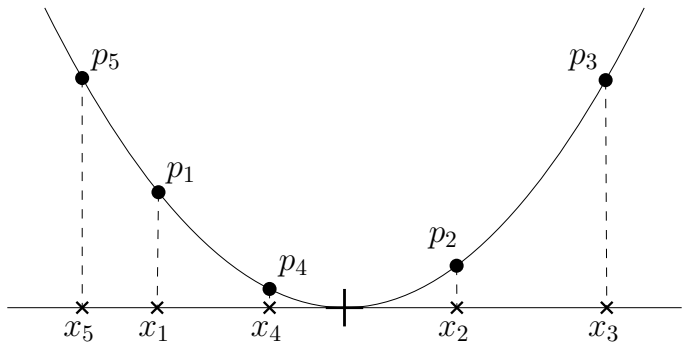
Betrachte eine Folge $(x_1, x_2, \ldots, x_n)$ von Zahlen in $\mathbb{R}$. Wir setzen $p_i = (x_i, x_i^2)$, $i = 1, 2, \ldots, n$ (vertikale Projektion von der x-Achse im $\mathbb{R}^2$ auf die Einheitsparabel $y = x^2$).



Aus der Folge der Ecken der konvexen Hülle von $P := \{p_1, p_2, \ldots, p_n\}$ ergibt sich (in linearer Zeit) auch die aufsteigend sortierte Reihenfolge der x_i's.

Untere Schranke für ConvexHull

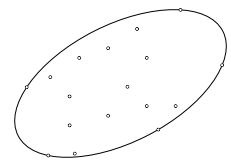
Betrachte eine Folge $(x_1, x_2, \ldots, x_n)$ von Zahlen in $\mathbb{R}$. Wir setzen $p_i = (x_i, x_i^2)$, $i = 1, 2, \ldots, n$ (vertikale Projektion von der x-Achse im $\mathbb{R}^2$ auf die Einheitsparabel $y = x^2$).

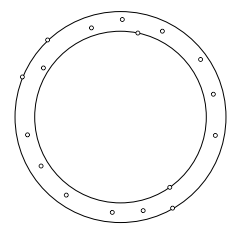


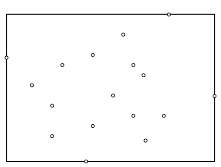
Aus der Folge der Ecken der konvexen Hülle von $P := \{p_1, p_2, \ldots, p_n\}$ ergibt sich (in linearer Zeit) auch die aufsteigend sortierte Reihenfolge der x_i's.

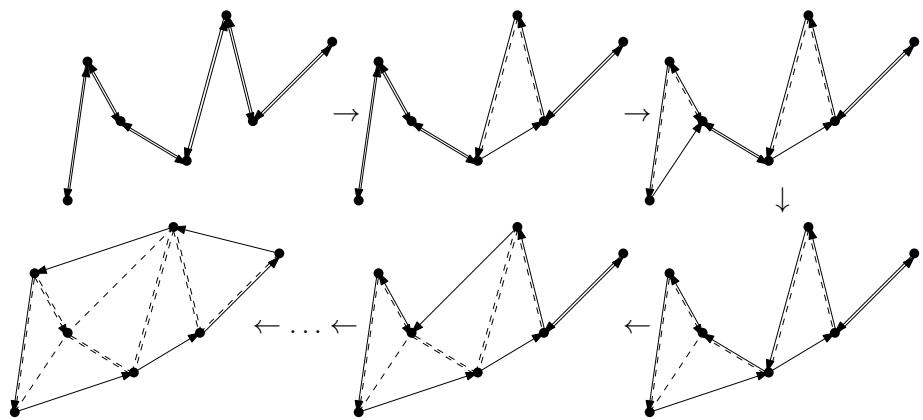
Wir haben eine sogenannte **Reduktion** gezeigt: Kann man ConvexHull in $t(n)$ lösen, so kann man in $t(n)+O(n)$ Zeit sortieren.

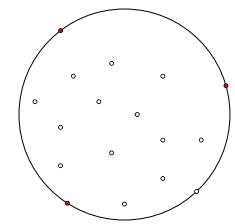
Konvexe Hülle: Lokales Verbessern

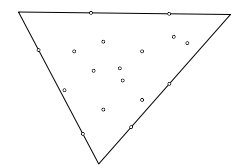


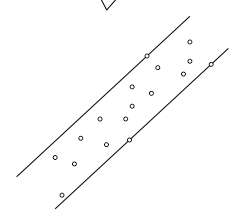




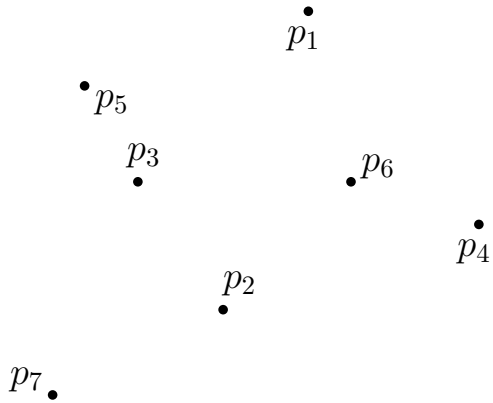




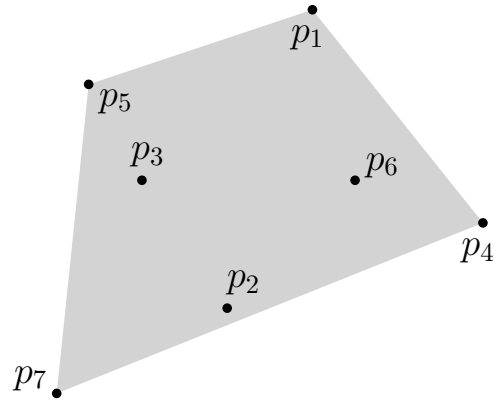




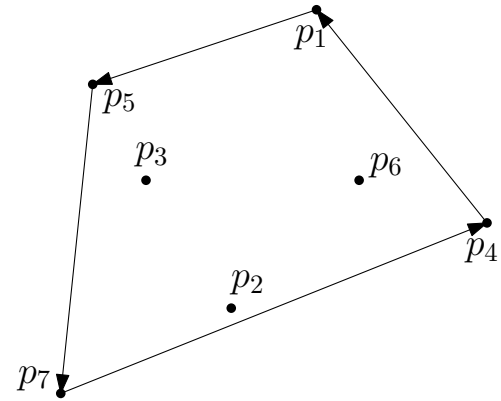
Problemstellung



Punktemenge P,



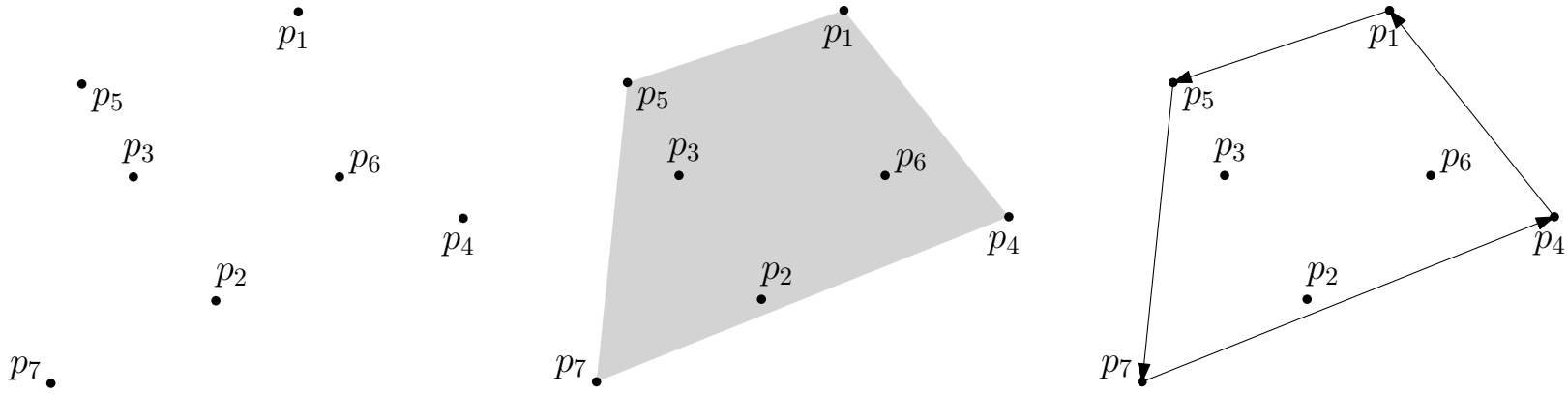
konvexe Hülle $\text{conv}(P)$,



Polygon (p_4, p_1, p_5, p_7).

ConvexHull-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme die Ecken des $\text{conv}(P)$ umrandenden Polygons, in der Reihenfolge gegen den Uhrseigersinn.

Problemstellung



Punktemenge P, konvexe Hülle $\text{conv}(P)$, Polygon (p_4, p_1, p_5, p_7).

ConvexHull-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme die Ecken des $\text{conv}(P)$ umrandenden Polygons, in der Reihenfolge gegen den Uhrseigersinn.

Vereinfachende Annahme: **Allgemeine Lage**, d.h. keine 3 Punkte auf einer gemeinsamen Geraden, keine 2 Pkt gleiche x-Koordinate.

Eine lokale Bedingung (lokal konvex)

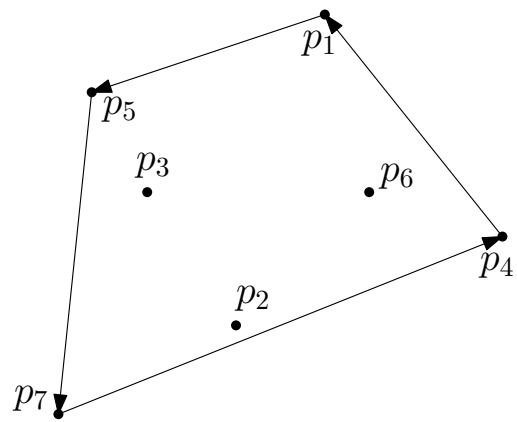
Ein Paar $qr \in P^2$, $q \neq r$, heisst **Randkante** von P, falls alle Punkte in $P \setminus \{q, r\}$ links von qr liegen.

Eine lokale Bedingung (lokal konvex)

Ein Paar $qr \in P^2$, $q \neq r$, heisst **Randkante** von P, falls alle Punkte in $P \setminus \{q, r\}$ links von qr liegen.

Lokale Prüfung in Polygon $(q_0, q_1, \ldots, q_{h-1})$.

$$\forall i, 1 \leq i \leq h\colon \; q_{i+1} \text{ links von } q_{i-1}q_i$$



Lokale Bedingung – Defizite

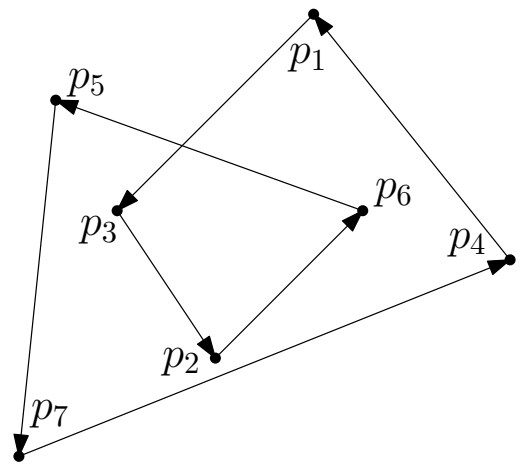
- $\{q_0, q_1, \ldots, q_{h-1}\}$ muss nicht Teilmenge von P sein.

Lokale Bedingung – Defizite

- $\{q_0, q_1, \ldots, q_{h-1}\}$ muss nicht Teilmenge von P sein.
- Das Polygon muss nicht alle anderen Punkte im Inneren haben.

Lokale Bedingung – Defizite

- $\{q_0, q_1, \ldots, q_{h-1}\}$ muss nicht Teilmenge von P sein.
- Das Polygon muss nicht alle anderen Punkte im Inneren haben.
- Das Polygon kann sich selbst kreuzen.



Idee: Wir beginnen mit einem nicht notwendigerweise konvexen Polygon, dass die Bedingungen oben nicht verletzt, und „**konvexifizieren**“ das sukzessive („**lokal Verbessern**“).

Lokal Verbessern

Gegeben $(q_0, q_1, \ldots, q_{k-1})$, falls

$$q_i \text{ links von } q_{i-1}q_{i+1} \text{ liegt}$$

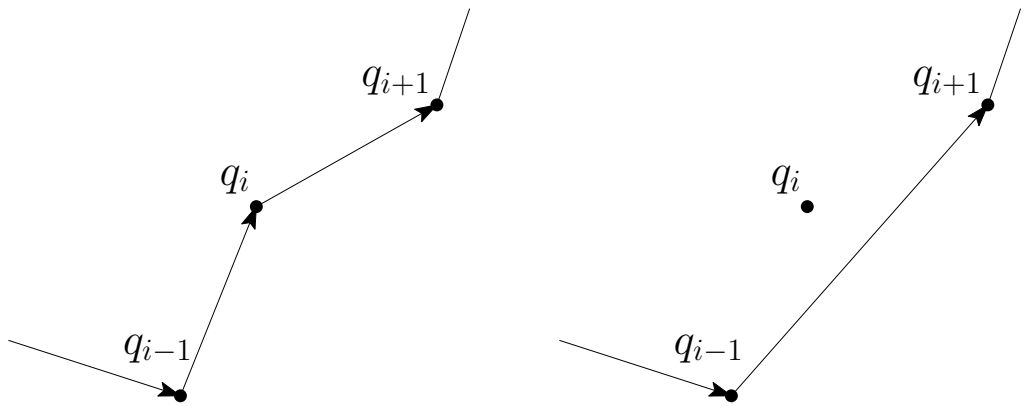
dann entferne q_i aus der Folge.

Lokal Verbessern

Gegeben $(q_0, q_1, \ldots, q_{k-1})$, falls

$$q_i \text{ links von } q_{i-1}q_{i+1} \text{ liegt}$$

dann entferne q_i aus der Folge.



Das Startpolygon

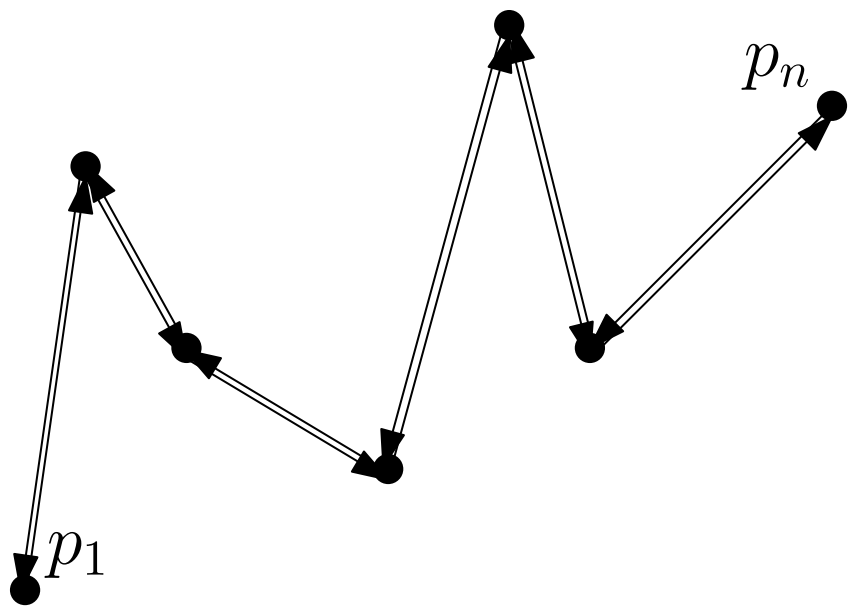
Sortiere P aufsteigend nach x-Koordinate: $(p_1, p_2, \ldots, p_n)$, und betrachte das Polygon

$$(p_1, p_2, \ldots, p_{n-1}, p_n, p_{n-1}, \ldots, p_2) \ ,$$

Das Startpolygon

Sortiere P aufsteigend nach x-Koordinate: $(p_1, p_2, \ldots, p_n)$, und betrachte das Polygon

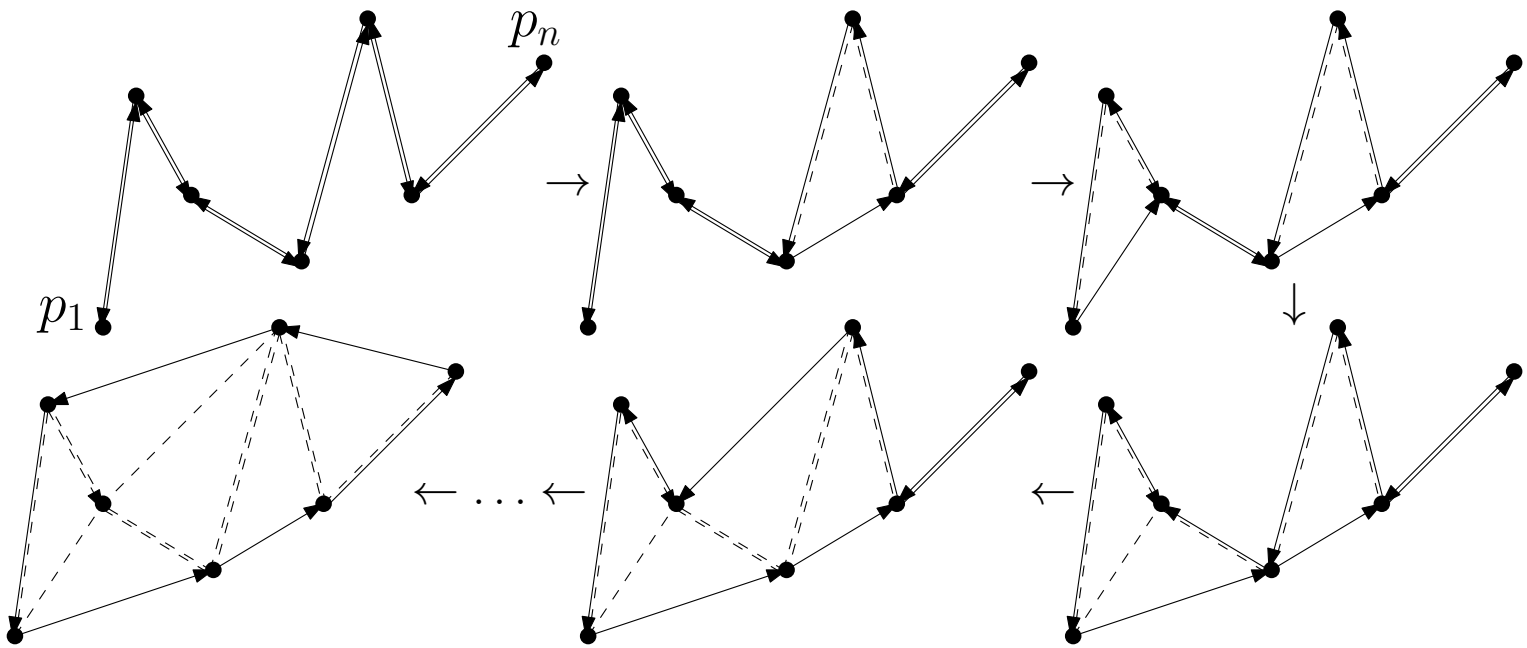
$$(p_1, p_2, \ldots, p_{n-1}, p_n, p_{n-1}, \ldots, p_2) \ ,$$



Das Startpolygon

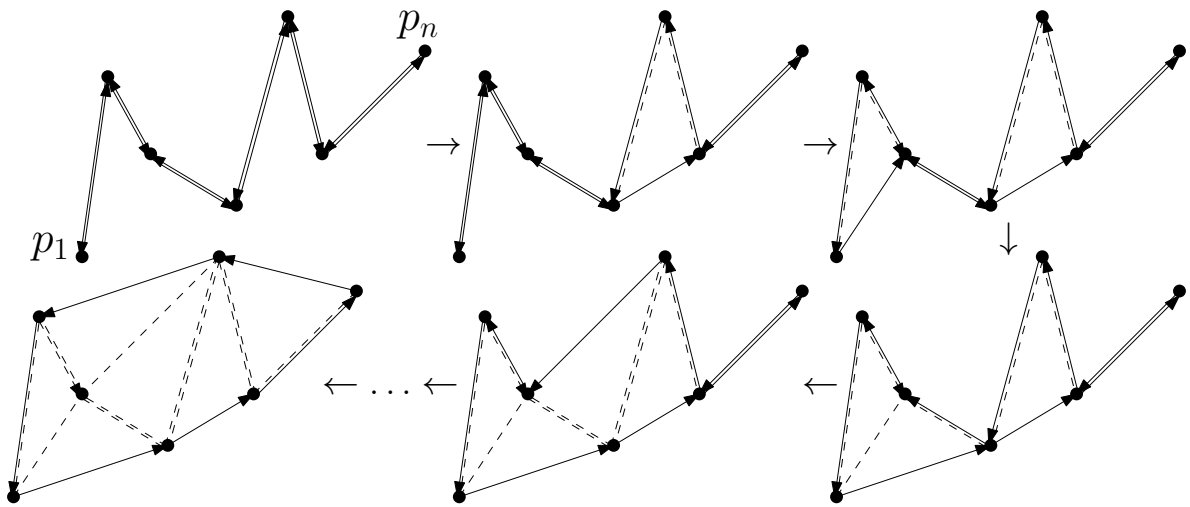
Sortiere P aufsteigend nach x-Koordinate: $(p_1, p_2, \ldots, p_n)$, und betrachte das Polygon

$$(p_1, p_2, \ldots, p_{n-1}, p_n, p_{n-1}, \ldots, p_2) \ ,$$



Invarianten während lokaler Verbesserungen

- Der Teilpolygonzug $(p_1, \ldots, p_n)$ ist x-monoton (von links nach rechts) und hat keinen Punkt in P unter sich.
- Der Teilpolygonzug $(p_n, \ldots, p_1)$ ist x-monoton (von rechts nach links) und hat keinen Punkt in P über sich.
- Der Teilpolygonzug $(p_1, \ldots, p_n)$ liegt nirgends über dem Teilpolygonzug $(p_n, \ldots, p_1)$.



Invarianten während lokaler Verbesserungen

- Der Teilpolygonzug $(p_1, \ldots, p_n)$ ist x-monoton (von links nach rechts) und hat keinen Punkt in P unter sich.
- Der Teilpolygonzug $(p_n, \ldots, p_1)$ ist x-monoton (von rechts nach links) und hat keinen Punkt in P über sich.
- Der Teilpolygonzug $(p_1, \ldots, p_n)$ liegt nirgends über dem Teilpolygonzug $(p_n, \ldots, p_1)$.

$\Rightarrow$ Das Polygon kann sich nie selber kreuzen und umschliesst alle Punkte. Folglich gilt: Lokal konvex $\Rightarrow$ Lösungspolygon.

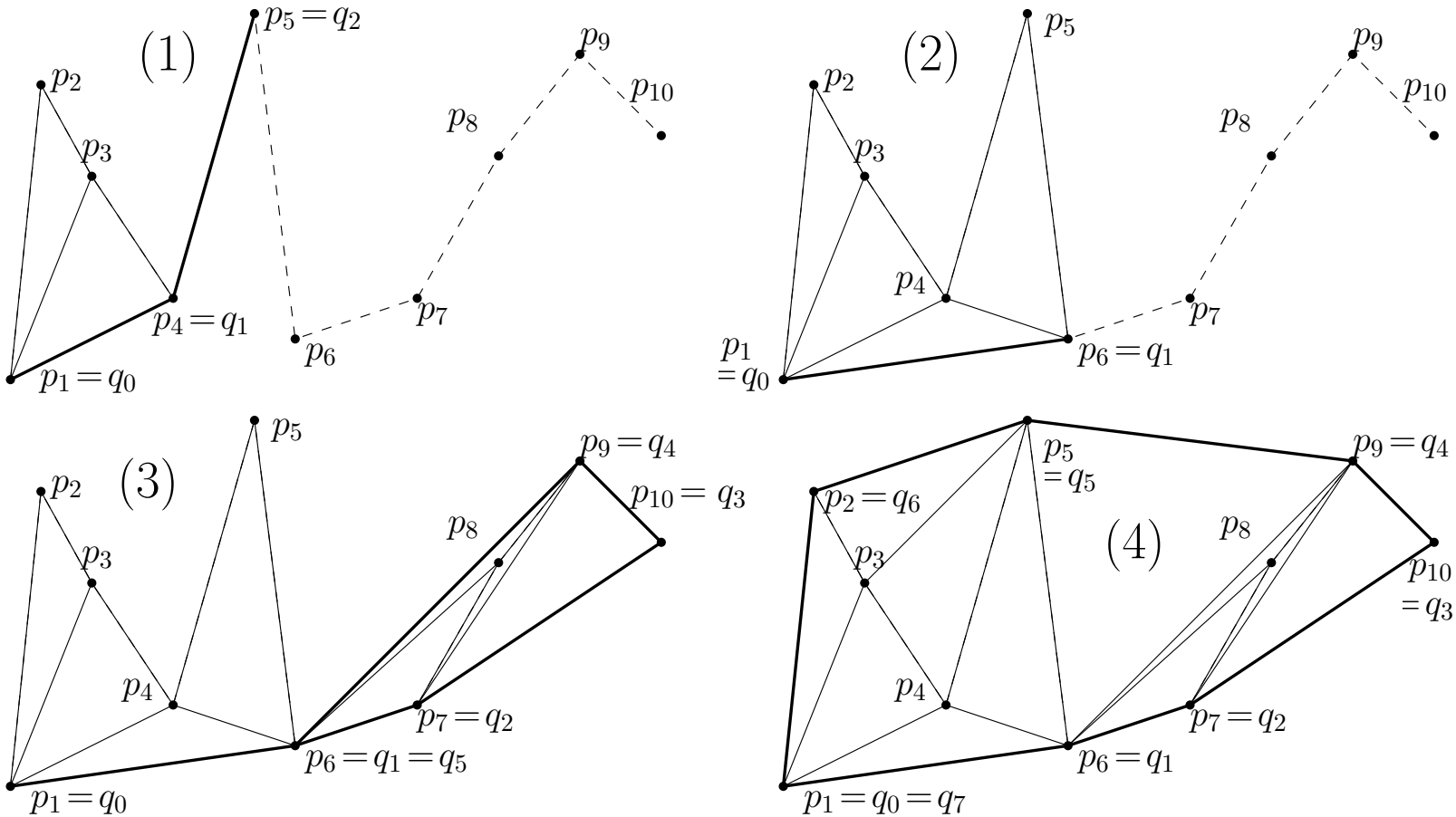
Bislang ist der Algorithmus **nicht-deterministisch**, d.h. wir können die lokalen Verbesserungsschritte in beliebiger Reihenfolge machen.

Der Algorithmus

LocalRepair$(p_1, p_2, \ldots, p_n)$ — $(p_1, p_2, \ldots, p_n)$ sortiert

1: $q_0 \leftarrow p_1$; $h \leftarrow 0$
2: **for** $i \leftarrow 2$ **to** n **do** ▷ unterer Rand, links nach rechts
3: **while** $h > 0$ und q_h links von $q_{h-1}p_i$ **do**
4: $h \leftarrow h - 1$
5: $h \leftarrow h + 1$; $q_h \leftarrow p_i$
6: ▷ $(q_0, \ldots, q_h)$ untere konvexe Hülle von $\{p_1, \ldots, p_i\}$
7: $h' \leftarrow h$
8: **for** $i \leftarrow n - 1$ **downto** 1 **do** ▷ oberer Rand, rechts nach links
9: **while** $h > h'$ und q_h links von $q_{h-1}p_i$ **do**
10: $h \leftarrow h - 1$
11: $h \leftarrow h + 1$; $q_h \leftarrow p_i$
12: **return** $(q_0, q_1, \ldots, q_{h-1})$

Der Algorithmus



Analyse

Wir beginnen mit einem Polygon mit $2(n-1)$ Ecken und haben am Ende h Ecken. Wir verbessern also genau $2(n-1)-h = O(n)$ Mal.

Analyse

Wir beginnen mit einem Polygon mit $2(n-1)$ Ecken und haben am Ende h Ecken. Wir verbessern also genau $2(n-1)-h = O(n)$ Mal.

Es gibt also $O(n)$ erfolgreiche Tests „q_h links von $q_{h-1}p_i$“, und für jeden Punkt p_i zwei erfolglose (einmal unten, und einmal oben).

Analyse

Wir beginnen mit einem Polygon mit $2(n-1)$ Ecken und haben am Ende h Ecken. Wir verbessern also genau $2(n-1)-h = O(n)$ Mal.

Es gibt also $O(n)$ erfolgreiche Tests „q_h links von $q_{h-1}p_i$“, und für jeden Punkt p_i zwei erfolglose (einmal unten, und einmal oben).

Der Algorithmus hat also nach dem anfänglichen Sortieren in $O(n \log n)$ eine Laufzeit von $O(n)$.

Analyse

Wir beginnen mit einem Polygon mit $2(n-1)$ Ecken und haben am Ende h Ecken. Wir verbessern also genau $2(n-1)-h = O(n)$ Mal.

Es gibt also $O(n)$ erfolgreiche Tests „q_h links von $q_{h-1}p_i$“, und für jeden Punkt p_i zwei erfolglose (einmal unten, und einmal oben).

Der Algorithmus hat also nach dem anfänglichen Sortieren in $O(n \log n)$ eine Laufzeit von $O(n)$.

Satz

Gegeben eine Folge $p_1, p_2, \ldots, p_n$ nach x-Koordinate sortierter Punkte in allgemeiner Lage in $\mathbb{R}^2$, berechnet der Algorithmus LocalRepair *die konvexe Hülle von $\{p_1, p_2, \ldots, p_n\}$ in Zeit $O(n)$.*

Analyse

Wir beginnen mit einem Polygon mit $2(n-1)$ Ecken und haben am Ende h Ecken. Wir verbessern also genau $2(n-1)-h = O(n)$ Mal.

Es gibt also $O(n)$ erfolgreiche Tests „q_h links von $q_{h-1}p_i$“, und für jeden Punkt p_i zwei erfolglose (einmal unten, und einmal oben).

Der Algorithmus hat also nach dem anfänglichen Sortieren in $O(n \log n)$ eine Laufzeit von $O(n)$.

Satz

Gegeben eine Folge $p_1, p_2, \ldots, p_n$ nach x-Koordinate sortierter Punkte in allgemeiner Lage in $\mathbb{R}^2$, berechnet der Algorithmus LocalRepair *die konvexe Hülle von $\{p_1, p_2, \ldots, p_n\}$ in Zeit $O(n)$.*

Wir haben argumentiert, dass Sortieren eine untere Schranke für das Problem ConvexHull ist. Der Algorithmus LocalRepair ist also **optimal**.

Anmerkungen

- Einbezug von Degeneriertheiten ist einfach (Sortieren lexikographisch, Duplikate nach Sortieren entfernen, „q_h links von $q_{h-1}p_i$“ adaptieren).
- Der Algorithmus ist numerisch robuster als JarvisWrap, kann nie eine unendliche Schleife laufen.

Anmerkungen

- Einbezug von Degeneriertheiten ist einfach (Sortieren lexikographisch, Duplikate nach Sortieren entfernen, „q_h links von $q_{h-1}p_i$“ adaptieren).
- Der Algorithmus ist numerisch robuster als JarvisWrap, kann nie eine unendliche Schleife laufen.
- Algorithmus liefert auch eine Triangulierung der Punkte.
- Lokale Verbesserung kann auch zur Berechnung „guter“ Triangulierungen (Delaunay Triangulierungen) verwendet werden.

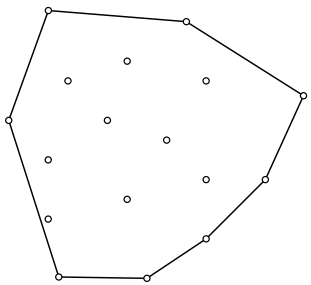
Anmerkungen

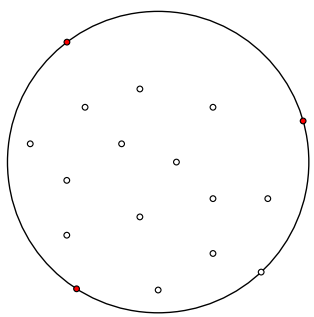
- Einbezug von Degeneriertheiten ist einfach (Sortieren lexikographisch, Duplikate nach Sortieren entfernen, „q_h links von $q_{h-1}p_i$“ adaptieren).
- Der Algorithmus ist numerisch robuster als JarvisWrap, kann nie eine unendliche Schleife laufen.
- Algorithmus liefert auch eine Triangulierung der Punkte.
- Lokale Verbesserung kann auch zur Berechnung „guter“ Triangulierungen (Delaunay Triangulierungen) verwendet werden.
- LocalRepair ist optimal, aber:

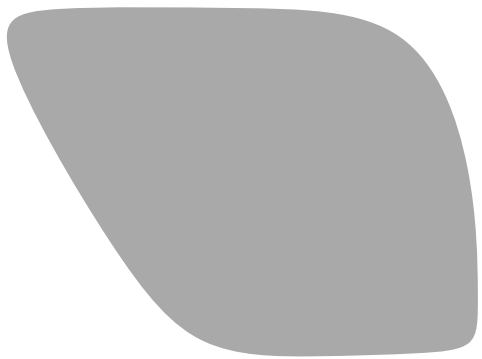
Anmerkungen

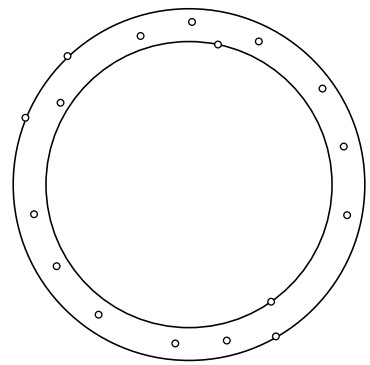
- Einbezug von Degeneriertheiten ist einfach (Sortieren lexikographisch, Duplikate nach Sortieren entfernen, „q_h links von $q_{h-1}p_i$“ adaptieren).
- Der Algorithmus ist numerisch robuster als JarvisWrap, kann nie eine unendliche Schleife laufen.
- Algorithmus liefert auch eine Triangulierung der Punkte.
- Lokale Verbesserung kann auch zur Berechnung „guter“ Triangulierungen (Delaunay Triangulierungen) verwendet werden.
- LocalRepair ist optimal, aber: (i) Es gibt auch einen $O(n \log h)$ Algorithmus und (ii) für Punkte zufällig aus Quadrat oder Kreischeibe gibt es einen erwartet $O(n)$ Zeit-Algorithmus.

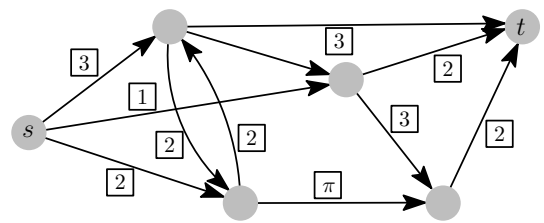
Konvexe Mengen











$\mathbb{R}^n$

Für M und N Mengen, ist

$$M^N \quad \text{die Menge aller Funktionen } N \to M.$$

(Wenn M und N endliche Mengen sind, dann gilt $\left|M^N\right| = |M|^{|N|}$.)

Insbesondere, ist $M^{[n]}$ die Menge der Folgen (bzw. Feld, Array) der Länge (bzw. Dimension) n mit Elementen aus M. Wir schreiben dafür auch oft einfach M^n.

$$\mathbb{R}^n \quad „\simeq“ \quad \mathbb{R}^{[n]} \quad „\simeq“ \quad \mathbb{R}^N \qquad \text{für } |N| = n$$

Beispiel – Fluss in Netzwerk

Ein **Netzwerk** ist ein Tupel $N = (V, A, c, s, t)$, mit V endliche Menge, $A \subseteq V^2 \setminus \{(v, v) \mid v \in V\}$, $s \in V$, $t \in V \setminus \{s\}$ und

$$c \in \mathbb{R}_0^{+A}.$$

Beispiel – Fluss in Netzwerk

Ein **Netzwerk** ist ein Tupel $N = (V, A, c, s, t)$, mit V endliche Menge, $A \subseteq V^2 \setminus \{(v, v) \mid v \in V\}$, $s \in V$, $t \in V \setminus \{s\}$ und

$$c \in \mathbb{R}_0^{+A}.$$

$$c \in \mathbb{R}_0^{+m}, \quad m := |A|$$
$$A = \{e_1, e_2, \ldots, e_m\}$$

Beispiel – Fluss in Netzwerk

Ein **Netzwerk** ist ein Tupel $N = (V, A, c, s, t)$, mit V endliche Menge, $A \subseteq V^2 \setminus \{(v, v) \mid v \in V\}$, $s \in V$, $t \in V \setminus \{s\}$ und

$$c \in \mathbb{R}_0^{+A}. \qquad c \in \mathbb{R}_0^{+m}, \quad m := |A| \\ A = \{e_1, e_2, \ldots, e_m\}$$

$f \in \mathbb{R}^A$ heisst **Fluss** von N, falls

$$\forall e \in A: \quad f(e) \geq 0 \text{ und } f(e) \leq c(e)$$

$$\forall v \in V \setminus \{s, t\}: \quad \sum_{e \in A} \sigma_{v,e} \cdot f(e) = 0$$

Beispiel – Fluss in Netzwerk

Ein **Netzwerk** ist ein Tupel $N = (V, A, c, s, t)$, mit V endliche Menge, $A \subseteq V^2 \setminus \{(v, v) \mid v \in V\}$, $s \in V$, $t \in V \setminus \{s\}$ und

$$c \in \mathbb{R}_0^{+A}. \qquad c \in \mathbb{R}_0^{+m}, \quad m := |A| \quad A = \{e_1, e_2, \ldots, e_m\}$$

$f \in \mathbb{R}^A$ heisst **Fluss** von N, falls

$$\forall e \in A: \quad f(e) \geq 0 \text{ und } f(e) \leq c(e)$$

$$\forall v \in V \setminus \{s, t\}: \quad \sum_{e \in A} \sigma_{v,e} \cdot f(e) = 0$$

wobei $\sigma_{v,e} := \begin{cases} -1 & e \text{ in } v \text{ eingehende Kante,} \\ 1 & e \text{ aus } v \text{ ausgehende Kante, und} \\ 0 & \text{sonst.} \end{cases}$

$$\left[\forall v \in V \setminus \{s, t\}: \sum_{u \in V: (u,v) \in A} f(u, v) = \sum_{u \in V: (v,u) \in A} f(v, u) \right]$$

Beispiel – Fluss in Netzwerk

Ein **Netzwerk** ist ein Tupel $N = (V, A, c, s, t)$, mit V endliche Menge, $A \subseteq V^2 \setminus \{(v, v) \mid v \in V\}$, $s \in V$, $t \in V \setminus \{s\}$ und

$$c \in \mathbb{R}_0^{+A}. \qquad c \in \mathbb{R}_0^{+m}, \quad m := |A| \quad A = \{e_1, e_2, \ldots, e_m\}$$

$f \in \mathbb{R}^A$ heisst **Fluss** von N, falls

$$\forall e \in A: \quad f(e) \geq 0 \text{ und } f(e) \leq c(e)$$

$$\forall v \in V \setminus \{s, t\}: \quad \sum_{e \in A} \sigma_{v,e} \cdot f(e) = 0$$

wobei $\sigma_{v,e} := \begin{cases} -1 & e \text{ in } v \text{ eingehende Kante,} \\ 1 & e \text{ aus } v \text{ ausgehende Kante, und} \\ 0 & \text{sonst.} \end{cases}$

Flüsse sind Punkte (bzw. Vektoren) im $\mathbb{R}^A$ „$\simeq$“ $\mathbb{R}^m$ mit gewissen Bedingungen.

Konvexe Mengen

Sei $n \in \mathbb{N}$.

- Für $v_0, v_1 \in \mathbb{R}^n$ sei

$$\overline{v_0 v_1} := \{(1-\lambda)v_0 + \lambda v_1 \mid \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1\} ,$$

  das v_0 und v_1 verbindende **Liniensegment**.

- Eine Menge $C \subseteq \mathbb{R}^n$ heisst **konvex**, falls

$$\forall v_0, v_1 \in C : \ \overline{v_0 v_1} \subseteq C .$$

Beobachtung

Der Schnitt konvexer Mengen ist eine konvexe Menge.

Gerade, Ebene, Hyperebene

Gerade im $\mathbb{R}^2$ $(a, b) \neq (0, 0)$

Für $(a, b, c) \in \mathbb{R}^3$ gegeben: $\{(x, y) \in \mathbb{R}^2 \mid ax + by = c\}$

Gerade, Ebene, Hyperebene

Gerade im $\mathbb{R}^2$ $(a, b) \neq (0, 0)$

Für $(a, b, c) \in \mathbb{R}^3$ gegeben: $\{(x, y) \in \mathbb{R}^2 \mid ax + by = c\}$

Beachte: Ist $(a', b', c') = \mu \cdot (a, b, c)$, $\mu \in \mathbb{R} \backslash \{0\}$, dann definieren (a', b', c') und (a, b, c) die gleiche Gerade.

Gerade, Ebene, Hyperebene

Gerade im $\mathbb{R}^2$ $(a, b) \neq (0, 0)$

Für $(a, b, c) \in \mathbb{R}^3$ gegeben: $\{(x, y) \in \mathbb{R}^2 \mid ax + by = c\}$

Ebene im $\mathbb{R}^3$ $(a, b, c) \neq (0, 0, 0)$

Für $(a, b, c, d) \in \mathbb{R}^4$ geg.: $\{(x, y, z) \in \mathbb{R}^3 \mid ax + by + cz = d\}$

Gerade, Ebene, Hyperebene

Gerade im $\mathbb{R}^2$ $\qquad (a, b) \neq (0, 0)$

Für $(a, b, c) \in \mathbb{R}^3$ gegeben: $\{(x, y) \in \mathbb{R}^2 \mid ax + by = c\}$

Ebene im $\mathbb{R}^3$ $\qquad (a, b, c) \neq (0, 0, 0)$

Für $(a, b, c, d) \in \mathbb{R}^4$ geg.: $\{(x, y, z) \in \mathbb{R}^3 \mid ax + by + cz = d\}$

$$\left.\begin{aligned} ax + by + cz &= d \\ ax' + by' + cz' &= d \end{aligned}\right\} \Rightarrow$$

$$a\big((1-\lambda)x + \lambda x'\big) + b\big((1-\lambda)y + \lambda y'\big) + c\big((1-\lambda)z + \lambda z'\big) =$$
$$(1-\lambda)\big(\underbrace{ax + by + cz}_{d}\big) + \lambda\big(\underbrace{ax' + by' + cz'}_{d}\big) = (1-\lambda)d + \lambda d = d$$

D.h. jede Ebene ist konvex.

Gerade, Ebene, Hyperebene

Gerade im $\mathbb{R}^2$ $(a, b) \neq (0, 0)$

Für $(a, b, c) \in \mathbb{R}^3$ gegeben: $\{(x, y) \in \mathbb{R}^2 \mid ax + by = c\}$

Ebene im $\mathbb{R}^3$ $(a, b, c) \neq (0, 0, 0)$

Für $(a, b, c, d) \in \mathbb{R}^4$ geg.: $\{(x, y, z) \in \mathbb{R}^3 \mid ax + by + cz = d\}$

Hyperebene im $\mathbb{R}^n$ $(a_1, \ldots, a_n) \neq (0, \ldots, 0)$

Für $(a_0, \ldots, a_n) \in \mathbb{R}^{n+1}$ geg.: $\{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \sum_{i=1}^{n} a_i x_i = a_0\}$

Beobachtung

Hyperebenen sind konvexe Mengen.
Schnitte von Hyperebenen sind konvexe Mengen.

Beispiel - Flusserhaltung

$f \in \mathbb{R}^A$ heisst **Fluss** von Netzwerk $N = (V, A, c, s, t)$, falls

$$\forall e \in A: \quad f(e) \geq 0 \text{ und } f(e) \leq c(e)$$

$$\forall v \in V \setminus \{s, t\} \quad \sum_{e \in A} \sigma_{v,e} \cdot f(e) = 0$$

↖ Flusserhaltung

Die Flusserhaltungsbedingungen sind Hyperebenen. Die Flüsse, die alle diese Flusserhaltungen erfüllen bilden also einen Schnitt von Hyperebenen und daher eine konvexe Menge.

Beispiel - Flusserhaltung

$f \in \mathbb{R}^A$ heisst **Fluss** von Netzwerk $N = (V, A, c, s, t)$, falls $f(e) = x_e$ für Zahlen x_e, $e \in A$, mit

$$\forall e \in A: \quad x_e \geq 0 \text{ und } x_e \leq c(e)$$

$$\forall v \in V \setminus \{s, t\} \quad \sum_{e \in A} \sigma_{v,e} \cdot x_e = 0$$

↖ Flusserhaltung

Die Flusserhaltungsbedingungen sind Hyperebenen. Die Flüsse, die alle diese Flusserhaltungen erfüllen bilden also einen Schnitt von Hyperebenen und daher eine konvexe Menge.

Halbebene, Halbräume

Halbebene im $\mathbb{R}^2$ $(a, b) \neq (0, 0)$

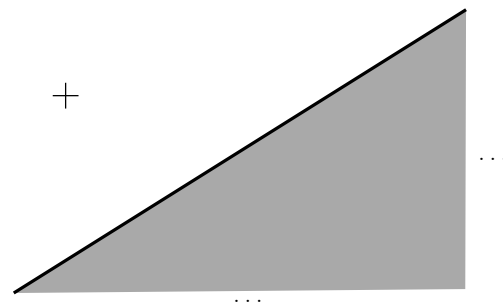
Für $(a, b, c) \in \mathbb{R}^3$ gegeben: $\{(x, y) \in \mathbb{R}^2 \mid ax + by \geq c\}$

Halbebene, Halbräume

Halbebene im $\mathbb{R}^2$ $(a, b) \neq (0, 0)$

Für $(a, b, c) \in \mathbb{R}^3$ gegeben: $\{(x, y) \in \mathbb{R}^2 \mid ax + by \geq c\}$

Beachte: Jeder Halbebene ist konvex.



Halbebene, Halbräume

Halbebene im $\mathbb{R}^2$ $(a, b) \neq (0, 0)$

Für $(a, b, c) \in \mathbb{R}^3$ gegeben: $\{(x, y) \in \mathbb{R}^2 \mid ax + by \geq c\}$

Halbraum im $\mathbb{R}^n$ $(a_1, \ldots, a_n) \neq (0, \ldots, 0)$

Für $(a_0, \ldots, a_n) \in \mathbb{R}^{n+1}$ geg.: $\{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \sum_{i=1}^{n} a_i x_i \geq a_0\}$

Beobachtung

Halbräume sind konvexe Mengen.
Schnitte von Halbräumen sind konvexe Mengen.

Beispiele – Schnitte von Halbräumen

- Intervall $[a, b]$ in $\mathbb{R}^1$: $\{x_1 \geq a\} \cap \{-x_1 \geq -b\}$, kurz:
$$\{a \leq x_1 \leq b\}$$

Beispiele – Schnitte von Halbräumen

- Intervall $[a, b]$ in $\mathbb{R}^1$: $\{x_1 \geq a\} \cap \{-x_1 \geq -b\}$, kurz:
$$\{a \leq x_1 \leq b\}$$
- Einheitsquadrat $[0, 1]^2$ in $\mathbb{R}^2$: $\{\underbrace{1 \cdot x_1 + 0 \cdot x_2 \geq 0}_{x_1 \geq 0}\} \cap \ldots$, kurz:
$$\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}$$

Beispiele – Schnitte von Halbräumen

- Intervall $[a, b]$ in $\mathbb{R}^1$: $\{x_1 \geq a\} \cap \{-x_1 \geq -b\}$, kurz:
$$\{a \leq x_1 \leq b\}$$
- Einheitsquadrat $[0, 1]^2$ in $\mathbb{R}^2$: $\{\underbrace{1 \cdot x_1 + 0 \cdot x_2 \geq 0}_{x_1 \geq 0}\} \cap \ldots$, kurz:
$$\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}$$
- Einheitswürfel $[0, 1]^3$ in $\mathbb{R}^3$:
$$\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, 0 \leq x_3 \leq 1\}$$

Beispiele – Schnitte von Halbräumen

- Intervall $[a, b]$ in $\mathbb{R}^1$: $\{x_1 \geq a\} \cap \{-x_1 \geq -b\}$, kurz:
$$\{a \leq x_1 \leq b\}$$
- Einheitsquadrat $[0, 1]^2$ in $\mathbb{R}^2$: $\{\underbrace{1 \cdot x_1 + 0 \cdot x_2 \geq 0}_{x_1 \geq 0}\} \cap \ldots$, kurz:
$$\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}$$
- Einheitswürfel $[0, 1]^3$ in $\mathbb{R}^3$:
$$\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, 0 \leq x_3 \leq 1\}$$
- (Einheits-) Hyperwürfel $[0, 1]^n$ in $\mathbb{R}^n$:
$$\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, \ldots, 0 \leq x_n \leq 1\}$$
(mit der Eckenmenge $\{0, 1\}^n$, 0-1-Folgen der Länge n)

Beispiele – Schnitte von Halbräumen

- Intervall $[a, b]$ in $\mathbb{R}^1$: $\{x_1 \geq a\} \cap \{-x_1 \geq -b\}$, kurz:
$$\{a \leq x_1 \leq b\}$$
- Einheitsquadrat $[0, 1]^2$ in $\mathbb{R}^2$: $\{\underbrace{1 \cdot x_1 + 0 \cdot x_2 \geq 0}_{x_1 \geq 0}\} \cap \ldots$, kurz:
$$\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}$$
- Einheitswürfel $[0, 1]^3$ in $\mathbb{R}^3$:
$$\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, 0 \leq x_3 \leq 1\}$$
- (Einheits-) Hyperwürfel $[0, 1]^n$ in $\mathbb{R}^n$:
$$\{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1, \ldots, 0 \leq x_n \leq 1\}$$
(mit der Eckenmenge $\{0, 1\}^n$, 0-1-Folgen der Länge n)
- Quader in $\mathbb{R}^n$:
$$\{a_1 \leq x_1 \leq b_1, a_2 \leq x_2 \leq b_2, \ldots, a_n \leq x_n \leq b_n\}$$

Beispiele – Schnitte von Halbräumen

- $\mathbb{R}^2$: $\{x_1 \geq 0\} \cap \{x_1 - x_2 \geq 0\} \cap \{x_2 \geq 1\} \cap \{x_1 + x_2 \geq \frac{1}{2}\}$,

  (nicht beschränkt!)

Beispiele – Schnitte von Halbräumen

- $\mathbb{R}^2$: $\{x_1 \geq 0\} \cap \{x_1 - x_2 \geq 0\} \cap \{x_2 \geq 1\} \cap \{x_1 + x_2 \geq \frac{1}{2}\}$,

  (nicht beschränkt!)

- ...

- Allgemein: Schnitte von Halbräumen nennt man **Polyeder**.

Beispiel – Zulässigkeit

$f \in \mathbb{R}^A$ heisst **Fluss** von Netzwerk $N = (V, A, c, s, t)$, falls

↙ Zulässigkeit

$$\forall e \in A: \quad f(e) \geq 0 \text{ und } f(e) \leq c(e)$$

$$\forall v \in V \setminus \{s, t\} \quad \sum_{e \in A} \sigma_{v,e} \cdot f(e) = 0$$

Die Zulässigkeitsbedingungen für $e \in A$ sind Halbräume. Flüsse bilden also einen Schnitt von $2m$ Halbräumen und $n - 2$ Hyperebenen im $\mathbb{R}^A$ ($n := |V|$, $m := |A|$).

Die Menge der Flüsse ist eine konvexe Menge im $\mathbb{R}^A$.

Ziel ist es in dieser konvexen Menge einen Punkt zu finden, der

$$\text{netoutflow}(s) := \sum_{e \in A} \sigma_{s,e} \cdot f(e)$$

(eine lineare Funktion) maximiert.

Beispiel - Zulässigkeit

$f \in \mathbb{R}^A$ heisst **Fluss** von Netzwerk $N = (V, A, c, s, t)$, falls $f(e) = x_e$ für Zahlen x_e, $e \in A$, mit

↙ Zulässigkeit

$$\forall e \in A: \quad x_e \geq 0 \text{ und } x_e \leq c(e)$$

$$\forall v \in V \setminus \{s, t\} \quad \sum_{e \in A} \sigma_{v,e} \cdot x_e = 0$$

Die Zulässigkeitsbedingungen für $e \in A$ sind Halbräume. Flüsse bilden also einen Schnitt von $2m$ Halbräumen und $n - 2$ Hyperebenen im $\mathbb{R}^A$ ($n := |V|$, $m := |A|$).

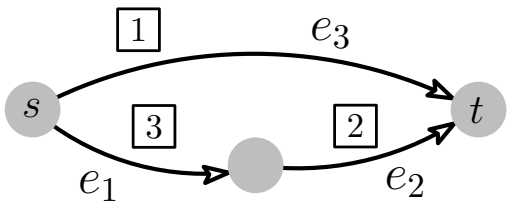
Die Menge der Flüsse ist eine konvexe Menge im $\mathbb{R}^A$.

Ziel ist es in dieser konvexen Menge einen Punkt zu finden, der

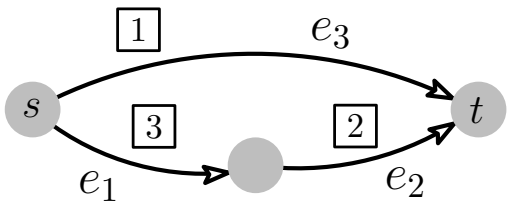
$$\text{netoutflow}(s) := \sum_{e \in A} \sigma_{s,e} \cdot x_e$$

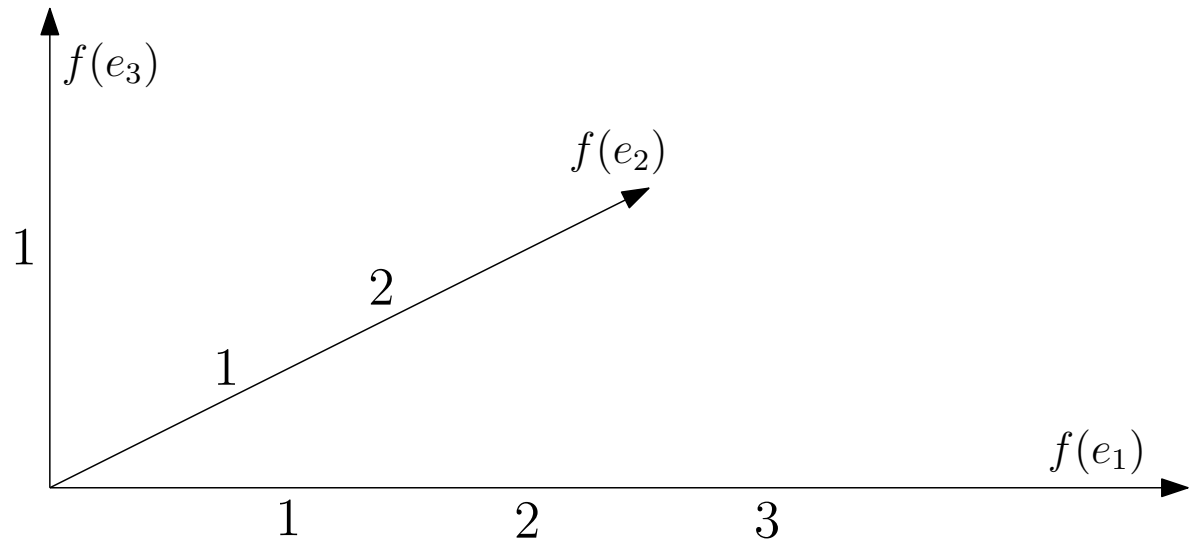
(eine lineare Funktion) maximiert.

Beispiel – Menge der Flüsse

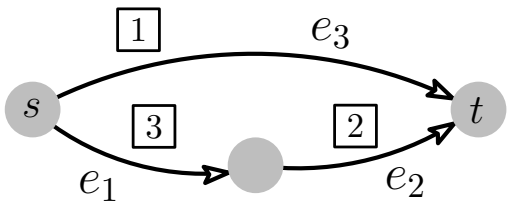


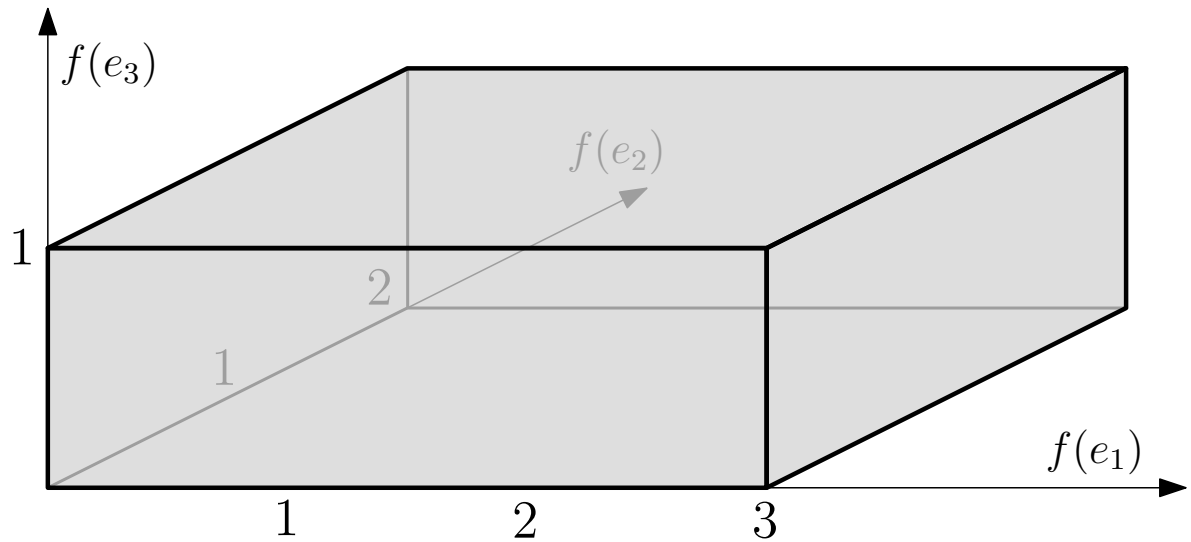
Beispiel – Menge der Flüsse



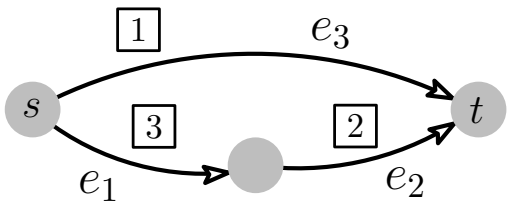


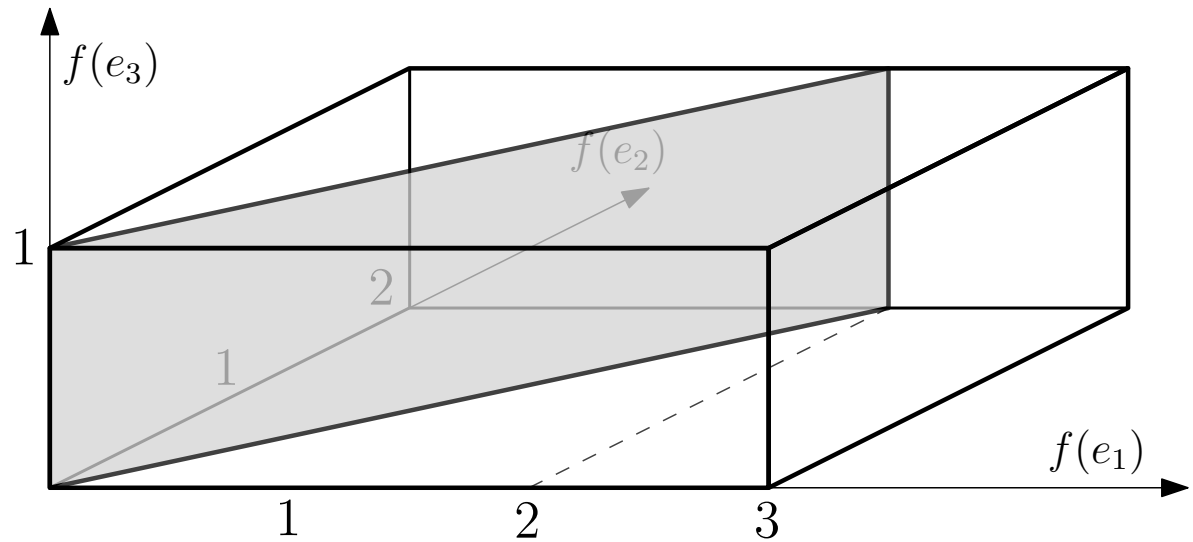
Beispiel – Menge der Flüsse



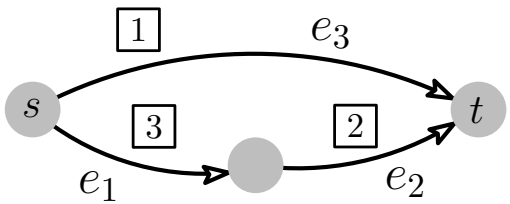


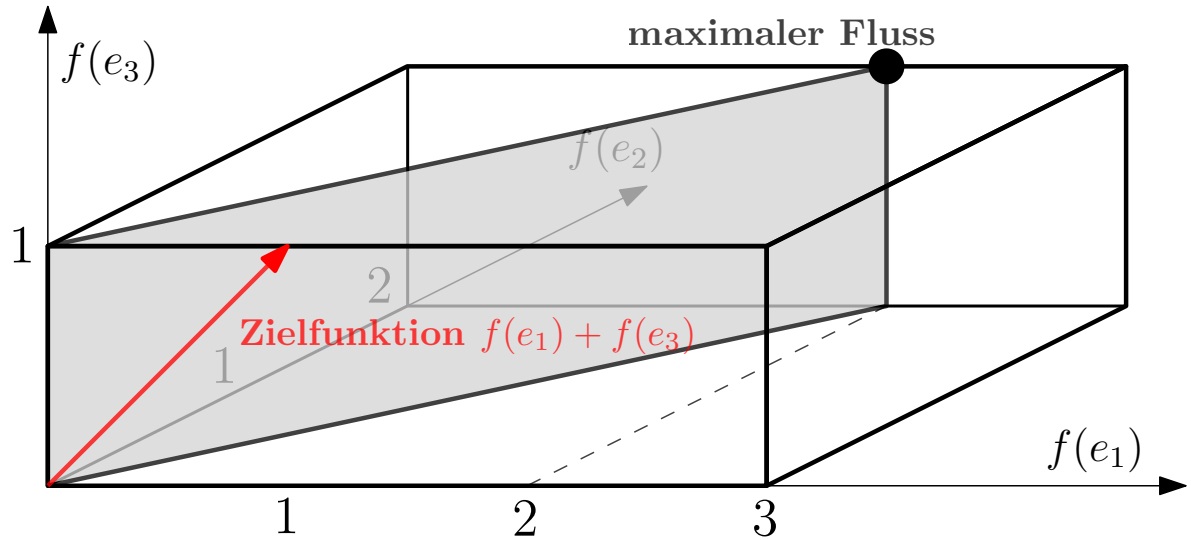
Beispiel – Menge der Flüsse



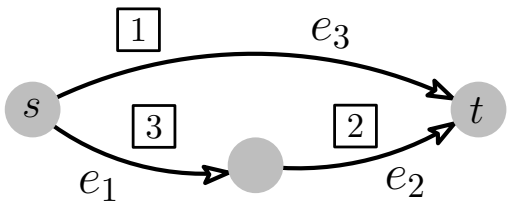


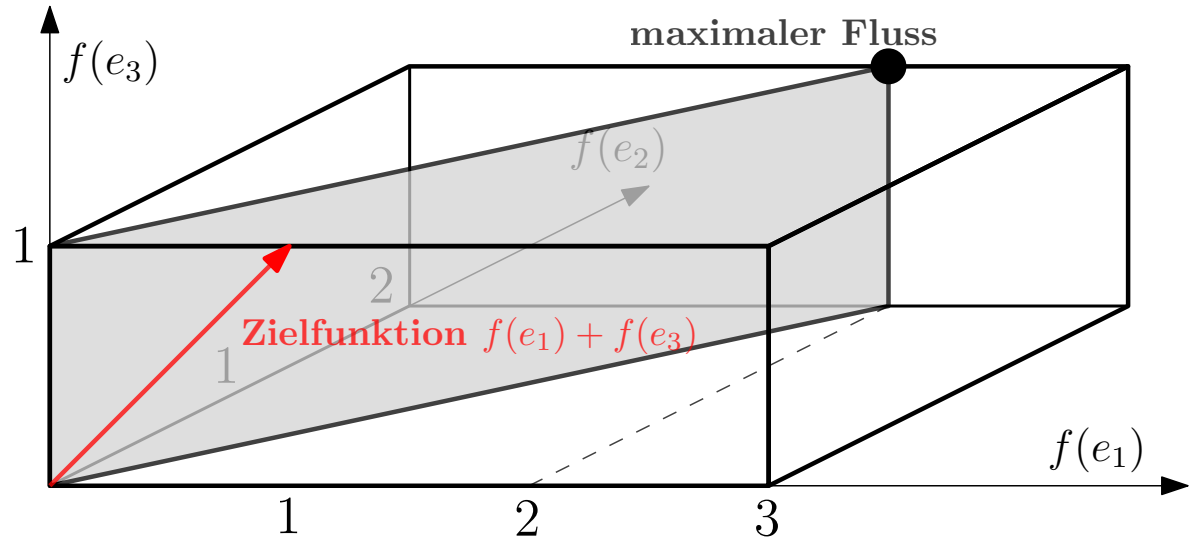
Beispiel – Menge der Flüsse





Beispiel – Menge der Flüsse





Ford-Fulkerson läuft am Rand der konvexen Menge aller Flüsse zum maximalen Fluss.

Kompakte Mengen

Hyperebenen und Halbräume (wie hier definiert) sind **abgeschlossene** Teilmengen des $\mathbb{R}^n$.

Kompakte Mengen

Hyperebenen und Halbräume (wie hier definiert) sind **abgeschlossene** Teilmengen des $\mathbb{R}^n$.

Der Durchschnitt endlich vieler abgeschlossener Mengen ist abgeschlossen.

Kompakte Mengen

Hyperebenen und Halbräume (wie hier definiert) sind **abgeschlossene** Teilmengen des $\mathbb{R}^n$.

Der Durchschnitt endlich vieler abgeschlossener Mengen ist abgeschlossen.

Die Menge aller Flüsse eines Netzwerks bildet daher eine abgeschlossene Menge im $\mathbb{R}^A$, zudem ist die Menge **beschränkt** (wegen $0 \leq f(e) \leq c(e)$). Das heisst, die Menge ist **kompakt**.

Kompakte Mengen

Hyperebenen und Halbräume (wie hier definiert) sind **abgeschlossene** Teilmengen des $\mathbb{R}^n$.

Der Durchschnitt endlich vieler abgeschlossener Mengen ist abgeschlossen.

Die Menge aller Flüsse eines Netzwerks bildet daher eine abgeschlossene Menge im $\mathbb{R}^A$, zudem ist die Menge **beschränkt** (wegen $0 \leq f(e) \leq c(e)$). Das heisst, die Menge ist **kompakt**.

Eine lineare Funktion nimmt in einer kompakten Menge das Minimum und Maximum an (folgt aus dem Satz vom Minimum und Maximum von Karl Weierstraß).

Kompakte Mengen

Hyperebenen und Halbräume (wie hier definiert) sind **abgeschlossene** Teilmengen des $\mathbb{R}^n$.

Der Durchschnitt endlich vieler abgeschlossener Mengen ist abgeschlossen.

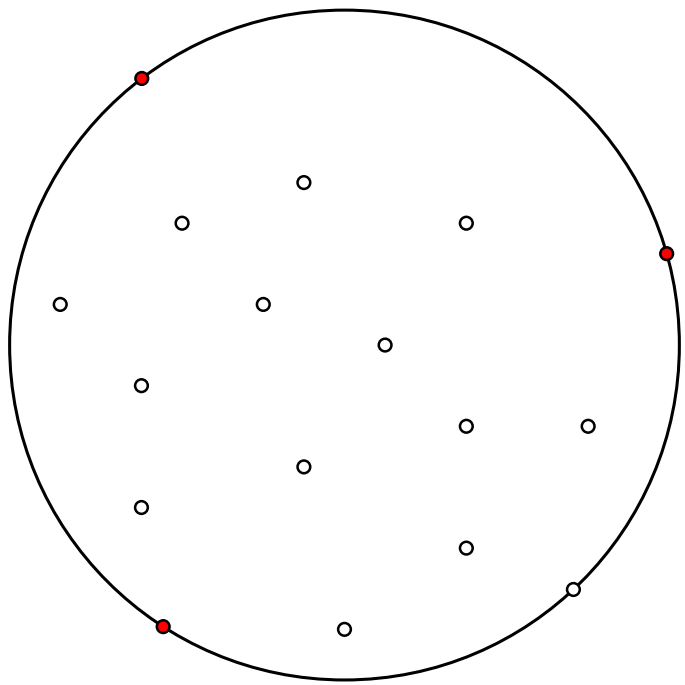
Die Menge aller Flüsse eines Netzwerks bildet daher eine abgeschlossene Menge im $\mathbb{R}^A$, zudem ist die Menge **beschränkt** (wegen $0 \leq f(e) \leq c(e)$). Das heisst, die Menge ist **kompakt**.

Eine lineare Funktion nimmt in einer kompakten Menge das Minimum und Maximum an (folgt aus dem Satz vom Minimum und Maximum von Karl Weierstraß).

Daraus folgt unmittelbar, dass jedes Netzwerk einen maximalen Fluss hat.

Beispiel – Kleinster umschliessender Kreis

SmallEnclDisk-Problem. Gegeben eine endliche Punktemenge $P \subseteq \mathbb{R}^2$, bestimme den Kreis kleinsten Radius, der P umschliesst.



Beispiel – Kleinster umschliessender Kreis

Ein Kreis ist bestimmt durch ein **Tupel** $K = (c_1, c_2, r) \in \underbrace{\mathbb{R}^2 \times \mathbb{R}_0^+}_{\text{Halbraum des } \mathbb{R}^3}$.

Der Kreis (des Kreis-Tupels K) umschliesst Punkt $p := (x_1, x_2)$ falls

$$(x_1 - c_1)^2 + (x_2 - c_2)^2 \leq r^2$$

Beispiel – Kleinster umschliessender Kreis

Ein Kreis ist bestimmt durch ein **Tupel** $K = (c_1, c_2, r) \in \underbrace{\mathbb{R}^2 \times \mathbb{R}_0^+}_{\text{Halbraum des } \mathbb{R}^3}$.

Der Kreis (des Kreis-Tupels K) umschliesst Punkt $p := (x_1, x_2)$ falls

$$(x_1 - c_1)^2 + (x_2 - c_2)^2 \leq r^2$$

Die Kreis-Tupel (c_1, c_2, r), deren Kreise $p = (x_1, x_2)$ umschliessen, bilden einen vertikalen Kegel (konvex) im $\mathbb{R}^3$ mit Spitze $(x_1, x_2, 0)$.

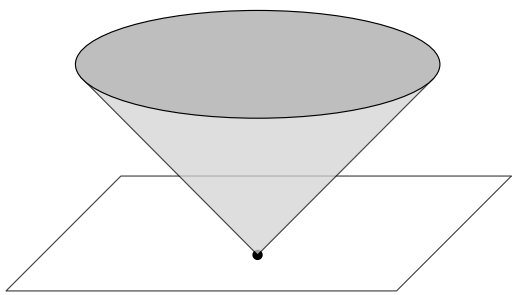
Beispiel – Kleinster umschliessender Kreis

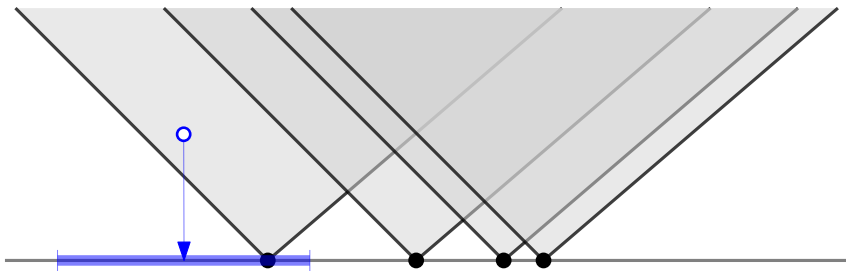
Ein Kreis ist bestimmt durch ein **Tupel** $K = (c_1, c_2, r) \in \underbrace{\mathbb{R}^2 \times \mathbb{R}_0^+}_{\text{Halbraum des } \mathbb{R}^3}$.

Der Kreis (des Kreis-Tupels K) umschliesst Punkt $p := (x_1, x_2)$ falls

$$(x_1 - c_1)^2 + (x_2 - c_2)^2 \leq r^2$$

Die Kreis-Tupel (c_1, c_2, r), deren Kreise $p = (x_1, x_2)$ umschliessen, bilden einen vertikalen Kegel (konvex) im $\mathbb{R}^3$ mit Spitze $(x_1, x_2, 0)$.





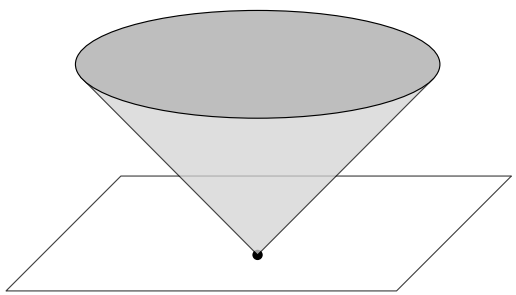
Beispiel – Kleinster umschliessender Kreis

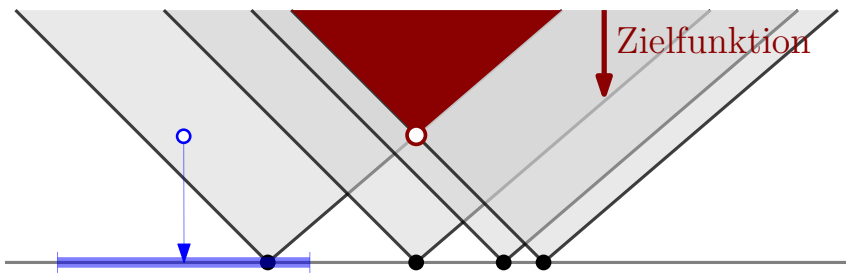
Ein Kreis ist bestimmt durch ein **Tupel** $K = (c_1, c_2, r) \in \underbrace{\mathbb{R}^2 \times \mathbb{R}_0^+}_{\text{Halbraum des } \mathbb{R}^3}$.

Der Kreis (des Kreis-Tupels K) umschliesst Punkt $p := (x_1, x_2)$ falls

$$(x_1 - c_1)^2 + (x_2 - c_2)^2 \leq r^2$$

Die Kreis-Tupel (c_1, c_2, r), deren Kreise $p = (x_1, x_2)$ umschliessen, bilden einen vertikalen Kegel (konvex) im $\mathbb{R}^3$ mit Spitze $(x_1, x_2, 0)$.





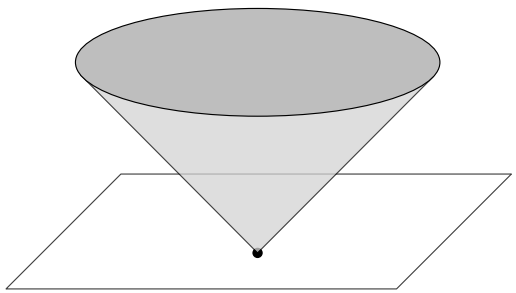
Beispiel – Kleinster umschliessender Kreis

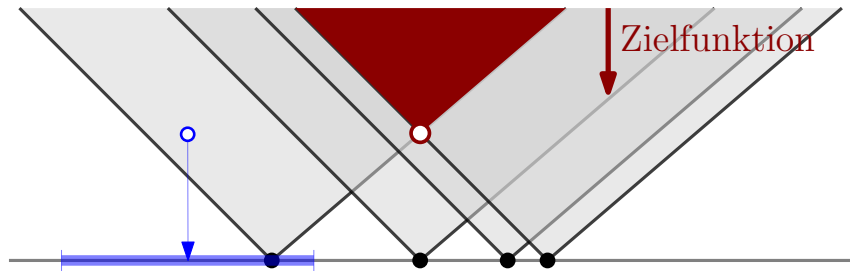
Ein Kreis ist bestimmt durch ein **Tupel** $K = (c_1, c_2, r) \in \underbrace{\mathbb{R}^2 \times \mathbb{R}_0^+}_{\text{Halbraum des } \mathbb{R}^3}$.

Der Kreis (des Kreis-Tupels K) umschliesst Punkt $p := (x_1, x_2)$ falls

$$(x_1 - c_1)^2 + (x_2 - c_2)^2 \leq r^2$$

Die Kreis-Tupel (c_1, c_2, r), deren Kreise $p = (x_1, x_2)$ umschliessen, bilden einen vertikalen Kegel (konvex) im $\mathbb{R}^3$ mit Spitze $(x_1, x_2, 0)$.





Unser Ziel ist es im Schnitt aller Kegel, eine konvexe Menge, einen Punkt (d.h. Kreis-Tupel) zu finden, das r minimiert. (Der tiefste Punkt im Schnitt ist immer durch höchstens drei Kegel bestimmt.)

Anmerkungen

- Konvexe Mengen zeichnen sich dadurch aus, dass man beim Optimieren nicht in einem lokalen nichtglobalen Optimum stecken bleiben kann.
- Es gibt reichhaltige Literatur und kommerzielle Software zur Optimierung in konvexen Mengen.