

Algorithms and Probability

Week 11

G08 - mkilic

15.V.2025

Overview

1. Minitest

2. Flüsse in Netzwerken

3. Flüsse in Netzwerken: Anwendungen

- 3.1 Bipartites Matching als Flussproblem
- 3.2 Kanten- und Knotendisjunkte Pfade
- 3.3 Bildsegmentierung

Roadmap

1. Graphentheorie

- Zusammenhang
- Kreise
- Matchings
- Färbungen

2. W'keitstheorie

- Bedingte W'keit
- Unabhängigkeit
- (mehrere) Zufallsvariablen
- Diskrete Verteilungen
- Abschätzen von W'keiten
- Randomisierte Algorithmen

3. Algorithmen

- Lange-Bunte Pfade
- MaxFlow
- MinCut
- Kleinster umschliessender Kreis
- Konvexe Hülle

Minitest

Passwort: flow

Flüsse - Einführung & Definitionen

Netzwerk

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* / *source*

$t \in V \setminus \{s\}$, die *Senke* / *target*, *sink*

$c : A \rightarrow \mathbb{R}_0^+$, die *Kapazitätsfunktion* / *capacity function*

Flüsse - Einführung & Definitionen

Der Fluss f

Gegeben sei ein Netzwerk $N = (V, A, c, s, t)$. Ein Fluss in N ist eine Funktion $f : A \rightarrow \mathbb{R}$ 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

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

⁰engl. Zulässigkeit: capacity constraint, Flusserhaltung: flow conservation constraint

Flüsse - Einführung & Definitionen

Der Wert eines Flusses

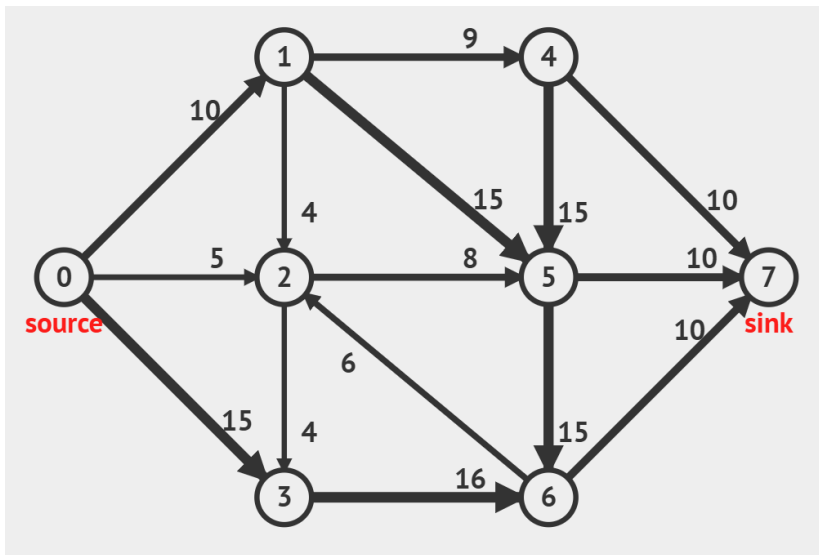
Der **Wert** eines Flusses f ist durch

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

definiert. Wir nennen f **ganzzahlig**, wenn $f(e) \in \mathbb{Z} \ \forall e \in A$.

⁰engl. Wert: value, ganzzahlig: integer-valued

Flussnetzwerk - Beispiel



Nettozufluss der Senke

Lemma 3.6

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

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

Also gilt $\text{netoutflow}(s) = \text{val}(f) = \text{netinflow}(t)$.

Netinflow(t) = Netoutflow(s) - Beweis

Beweis:

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

□

Schnitte

Schnitte

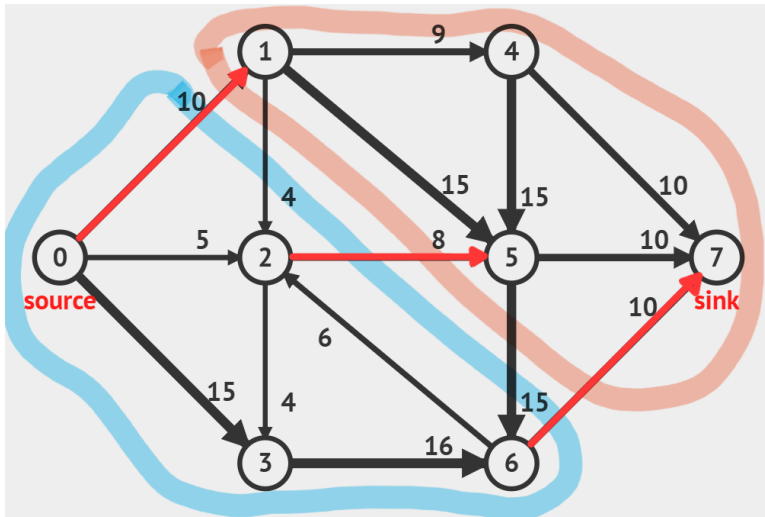
Ein *s-t-Schnitt* für ein Netzwerk (V, A, c, s, t) ist eine Partition (S, T) von V (d. h. $S \cup T = V$ und $S \cap T = \emptyset$) mit $s \in S$ und $t \in T$. Die *Kapazität* eines *s-t-Schnitts* (S, T) ist durch

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

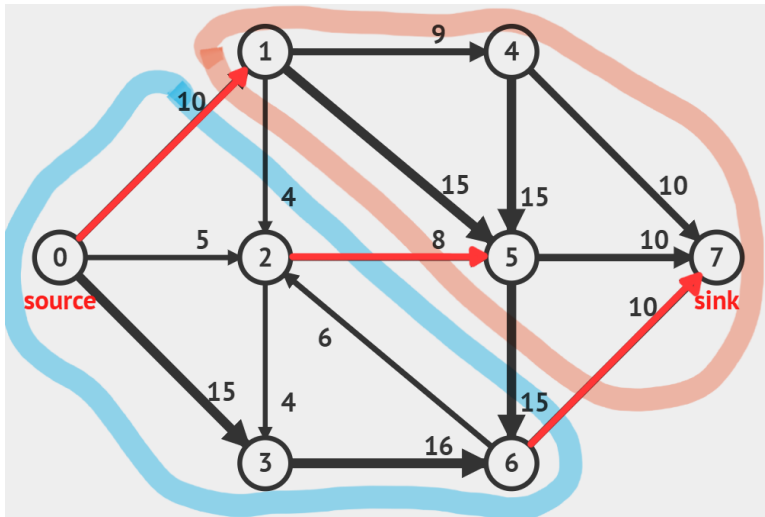
definiert.

- Die Kapazität eines Schnittes (S, T) ignoriert die Kanten von T nach S !
Nur Einweg!

Schnitte - Beispiel



Schnitte - Beispiel



$$\text{cap}(S, T) = 28$$

Schnitte und Flüsse

val(f) vs. *cap(S, T)*

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

$$val(f) \leq cap(S, T)$$

$val(f) \leq cap(S, T)$ - Beweis

Sei (U, W) eine Partition:

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

Jetzt behaupten wir

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

- (i) Wenn man nur die Flüsse für die Knoten in S berücksichtigt, dann bleiben am Ende nur die Kanten die nur ihren Anfangspunkt in S haben, oder nur ihren Endpunkt in S haben. In anderen Worten bleiben nur die Kanten die über den S - T -Schnitt verlaufen.
- (ii) f ist nichtnegativ
- (iii) $f(u, v) \leq c(u, v)$ für alle $(u, v) \in A$

⁰Formaler Beweis im Skript s. 176-177

Maxflow-Mincut Theorem

Maxflow-Mincut Theorem

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

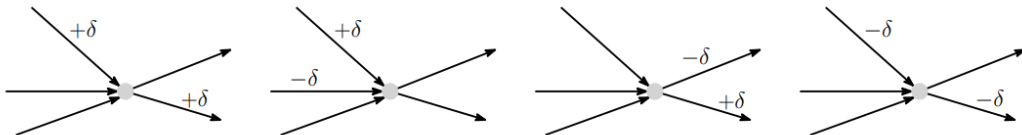
$$\max_{f \text{ Fluss in } N} \text{val}(f) = \min_{(S,T) \text{ s-t-Schnitt in } N} \text{cap}(S, T)$$

Besser weil:

- Es gibt nur endlich viele s - t -Schnitte.
- Ein minimaler Schnitt existiert immer

Das Restnetzwerk und Augmentierende Pfade

Grundidee: beginne mit einem beliebigen Fluss, dann verbessere den Fluss.
Wir haben 4 Tricks um lokale Veränderungen zu machen, die die Flusserhaltung erhalten.



Das Restnetzwerk und Augmentierende Pfade

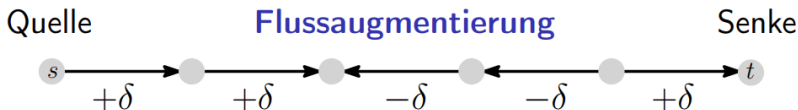
Wir betrachten Pfade von Quelle zu Senke. Lass P ein solcher Pfad sein:

Nehmen wir an, dass wir auf jeder vorwärts gerichteten Kante den Fluss um δ erhöhen können, ohne die Zulässigkeit zu verletzen, und auf jeder rückwärts gerichteten Kante den Fluss um δ verkleinern können, ohne dass dieser negativ wird.

Das Restnetzwerk und Augmentierende Pfade

Wir betrachten Pfade von Quelle zu Senke. Lass P ein solcher Pfad sein:

Nehmen wir an, dass wir auf jeder vorwärts gerichteten Kante den Fluss um δ erhöhen können, ohne die Zulässigkeit zu verletzen, und auf jeder rückwärts gerichteten Kante den Fluss um δ verkleinern können, ohne dass dieser negativ wird.



Wir können den Fluss verbessern ohne die Flusserhaltung zu verletzen.

Das Restnetzwerk und Augmentierende Pfade

Wenn wir wieder und wieder augmentieren dann erwarten wir dass wir in maximalen Flüssen stecken bleiben. Das muss man beweisen!

Man erreicht einen maximalen Fluss nicht immer in endlich vielen Schritten. Das gilt nur wenn die Kapazitäten **rational** sind!

Das Restnetzwerk und Augmentierende Pfade

Für Vereinfachung nehmen wir an dass wir in N keine entgegen gerichtete Kanten haben.

Restnetzwerk

Sei $N = (V, A, c, s, t)$ ein Netzwerk ohne entgegen gerichtete Kanten 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 auch 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. Nur Kanten wie in (1) und (2) beschrieben finden sich in A_f .

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

Charakterisierung Maximaler Fluss

Charakterisierung Maximaler Fluss **Satz 3.11**

Sei N ein Netzwerk ohne entgegen gerichtete Kanten.

Ein Fluss f ist maximal



Es gibt keinen gerichteten s - t -Pfad im Restnetzwerk N_f

Für jeden maximalen Fluss f gibt es einen s - t -Schnitt (S, T) mit

$$val(f) = cap(S, T)$$

Beweis: Satz 3.11

Beweisstruktur:

f maximal

(per contraposition, falls ein s-t-Pfad in N_f könnten wir f verbessern)

$\Rightarrow$ es gibt im N_f keinen gerichteten s-t-Pfad

(Schnitt mit $S =$ von s erreichbare Knoten in (V, A_f) und $T = V \setminus S$)

$\Rightarrow$ es gibt einen s-t-Schnitt (S, T) mit $val(f) = cap(S, T)$

(wir wissen $val(f) \leq cap(S, T)$. Wenn $val(f) = cap(S, T)$ ist dies maximal)

$\Rightarrow f$ maximal



Ford-Fulkerson

FORD-FULKERSON(V, A, c, s, t)

- 1: $f \leftarrow 0$ ▷ Fluss konstant 0
 - 2: **while** $\exists$ s - t -Pfad P in (V, A_f) **do**
 - 3: Erhöhe den Fluss entlang P ▷ wie unten
 - 4: **end while**
 - 5: **return** f ▷ maximaler Fluss = 0
-

▷ Erhöhung von f zu einem Fluss f' ist wie folgt:

$$f'(e) := \begin{cases} f(e) + \varepsilon & \text{falls } e \text{ auf } P, \\ f(e) - \varepsilon & \text{falls } e^{\text{opp}} \text{ auf } P, \text{ und} \\ f(e) & \text{sonst.} \end{cases}$$

Ford-Fulkerson Laufzeitanalyse

- Es ist möglich, dass der Algorithmus für irrationalen Kapazitäten nicht terminiert.
- Der Algorithmus terminiert immer für ganzzahligen Kapazitäten.
- Sei $U \in \mathbb{N}$ eine obere Schranke für die auftretende Kapazitäten, sei n die Anzahl der Knoten und m die Anzahl der Kanten.
- Dann kann kein Fluss Wert grösser als nU haben wegen
$$val(f) \leq cap(\{s\}, V \setminus \{s\}) \leq nU$$
- Also terminiert Ford-Fulkerson in höchstens nU Runden. Wir müssen in jeder Runde das Restnetzwerk aktualisieren und einen neuen gerichteten s - t -Pfad in N_f suchen. Das geht in $\mathcal{O}(n)$
- Insgesamt haben wir eine Laufzeit von $\mathcal{O}(mnU)$ falls alle Kapazitäten ganzzahlig ist.

Weitere Algorithmen

- **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.

Bipartites Matching als Flussproblem

Lemma 3.15

Die maximale Grösse eines Matchings im bipartiten Graph G ist gleich dem Wert eines maximalen Flusses im Netzwerk N . i.e.

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

Wir können aus einem ganzzahligen Fluss das Matching M herausfinden und aus ein Matching M den Fluss in N_G mit $\text{val}(f_M) = |M|$.

Formale Darstellung

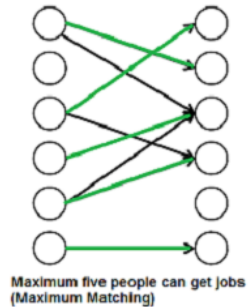
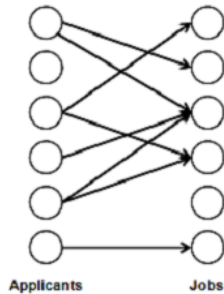
Wir haben: $G = (V, E)$ ein bipartiter Graph. D.h. Wir haben eine Partition von (U, W) for V , sodass $E \subseteq \{\{u, w\} \mid u \in U, w \in W\}$.

Wir definieren ein Netzwerk $N = (V \uplus \{s, t\}, A, c, s, t)$

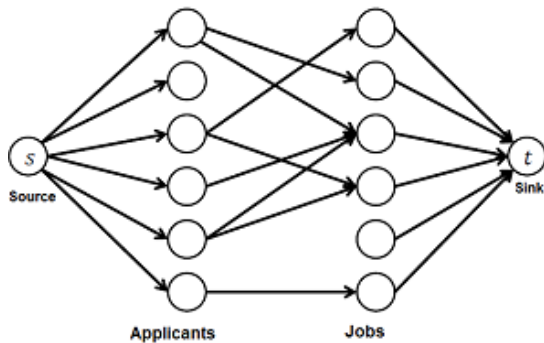
- Wir haben zwei neue Knoten die die Rolle von der Quelle und der Senke SPIELEN.
- Die Kapazitätsfunktion c ist konstant 1.
- s hat Kanten zu allen Knoten in U .
- t hat eingehende Kanten aus allen Knoten in W

$$A := (\{s\} \times U) \cup \{(u, w) \in U \times W \mid \{u, w\} \in E\} \cup (W \times \{t\})$$

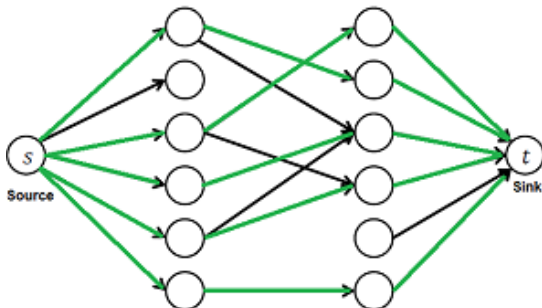
Beispiel: Matching Problem



Beispiel: Graph Konstruktion



Beispiel: Max Flow



The maximum flow from source to sink is five units. Therefore, maximum five people can get jobs.

Kanten- und Knotendisjunkte Pfade

Schritt 1 – Ungerichteten Graph in Netzwerk umwandeln

- Ausgangspunkt: Ungerichteter Graph $G = (V, E)$
- Für jede Kante $\{x, y\} \in E$:
 - Ersetze sie durch zwei gerichtete Kanten: (x, y) und (y, x)
 - Gib beiden Kanten Kapazität 1
- So entsteht ein gerichteter Graph mit Kapazitäten, der als Netzwerk genutzt werden kann.

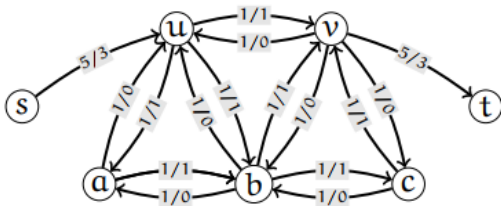
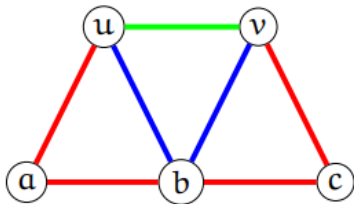
Schritt 2 – Quellen- und Senkenknoten einführen

- Füge zwei neue Knoten s und t hinzu
- Füge Kanten hinzu:
 - (s, u) mit Kapazität $n = |V|$
 - (v, t) mit Kapazität $n = |V|$
- Ziel: Ermögliche beliebig viele Wege von u nach v , beschränkt nur durch Kapazitäten in G

Warum funktioniert das?

- Alle Kanten im Netzwerk haben Kapazität 1
- Maximaler Fluss von s nach t entspricht der maximalen Anzahl **kanten-disjunkter** u - v -Pfade
- Warum?
 - Jeder Pfad nutzt disjunkte Kanten (wegen Kapazität 1)
 - Rückfluss (z.B. $f(x, y) = f(y, x) = 1$) kann durch Zyklusreduktion entfernt werden

Wie sieht es aus? - Beispiel Graph



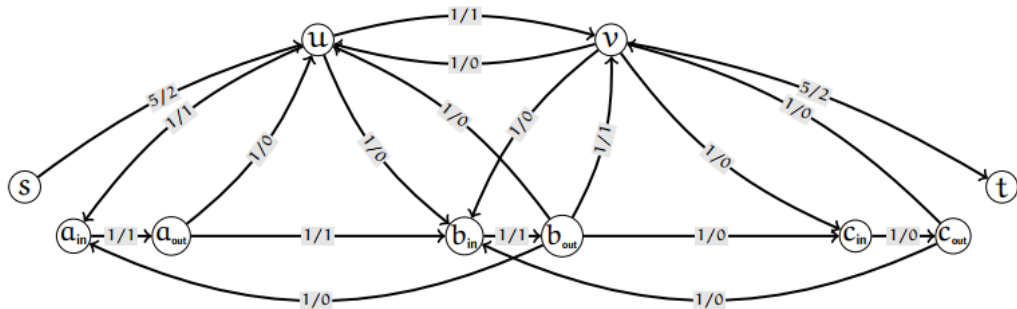
Knoten-disjunkte Pfade

- Etwas trickreicher: Um **knoten-disjunkte** Pfade zu zählen, wandle den Graphen wie folgt um:
 - Ersetze jeden Knoten $v \in V \setminus \{u, v\}$ durch zwei Knoten v_{in} und v_{out}
 - Verbinde $v_{\text{in}} \rightarrow v_{\text{out}}$ mit Kapazität 1
 - Ersetze jede Kante $\{x, y\} \in E$ durch zwei gerichtete Kanten:

$$x_{\text{out}} \rightarrow y_{\text{in}}, \quad y_{\text{out}} \rightarrow x_{\text{in}}$$

- So erzwingst du, dass kein Pfad denselben Knoten mehrfach benutzt.

Knotendisjunkte Pfade - Graphkonstruktion



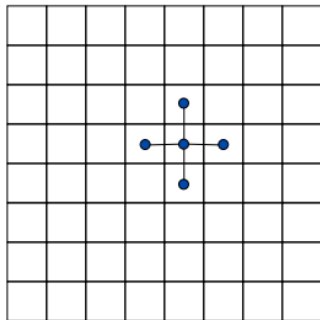
Bildsegmentierung mit Flüssen

Wir wollen für jedes Pixel entscheiden ob es im Hintergrund oder im Vordergrund steht. Wir haben ein Gitter und jedes Pixel hat ungerichtete Kanten zu den benachbarten Pixeln. (darüber, darunter, links und rechts). Wir haben also einen Graph (P, E) .

- $\alpha_p \geq 0$ ist die 'likelihood' dass $p \in A$
- $\beta_p \geq 0$ ist die 'likelihood' dass $p \in B$
- $p_{\{ij\}} \geq 0, \{i, j\} \in E$ ist Trennungsstrafe, also Strafe wenn einer der zwei Pixeln i, j als Vordergrund und das Andere als Hintergrund klassifiziert wird.

Wir wollen den Qualitätsfunktion maximieren:

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



Qualitätsfunktion Umformulieren

$$\begin{aligned}\arg \max_{A,B} q(A, B) &= \arg \max_{A,B} \sum_{p \in A} \alpha_p + \sum_{p \in B} \beta_p - \sum_{e \in E, |e \cap A|=1} \gamma_e \\ &= \arg \max_{A,B} \sum_{p \in P} (\alpha_p + \beta_p) - \sum_{p \in A} \beta_p - \sum_{p \in B} \alpha_p - \sum_{e \in E, |e \cap A|=1} \gamma_e \\ &= \arg \min_{A,B} q'(A, B) = \arg \min_{A,B} \sum_{p \in A} \beta_p + \sum_{p \in B} \alpha_p + \sum_{e \in E, |e \cap A|=1} \gamma_e\end{aligned}$$

Wir wollen den letzten Ausdruck minimieren. Dazu konstruieren wir einen Graphen.

Netzwerkaufbau für Bildsegmentierung

Gegeben: Menge von Pixeln P

Wir erstellen ein gerichtetes Netzwerk $N = (V, A)$ mit:

- Knotenmenge $V = P \cup \{s, t\}$, wobei s Quelle und t Senke ist
- Für jedes Pixel $p \in P$:
 - Kante (s, p) mit Kapazität α_p
 - Kante (p, t) mit Kapazität β_p
- Für jede ungerichtete Kante $\{p, p'\} \in E$:
 - Zwei gerichtete Kanten (p, p') und (p', p) mit Kapazität γ_e

Kapazität eines s - t -Schnitts

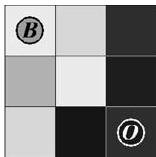
Ein Schnitt (S, T) im Netzwerk N trennt s von t . Sei:

$$A := S \setminus \{s\}, \quad B := T \setminus \{t\}$$

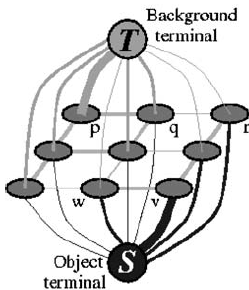
Die Schnittkapazität $cap(S, T)$ ergibt sich durch:

- **Quelle zu Pixeln:** $\sum_{p \in B} \alpha_p$
- **Pixel zu Senke:** $\sum_{p \in A} \beta_p$
- **Pixel-Pixel-Kanten:** $\sum_{(p,p') \in A \times B} \gamma_{(p,p')}$

Bildsegmentierung mit Flüssen - Beispielgraph



(a) Image with seeds.



Bedeutung für Segmentierung

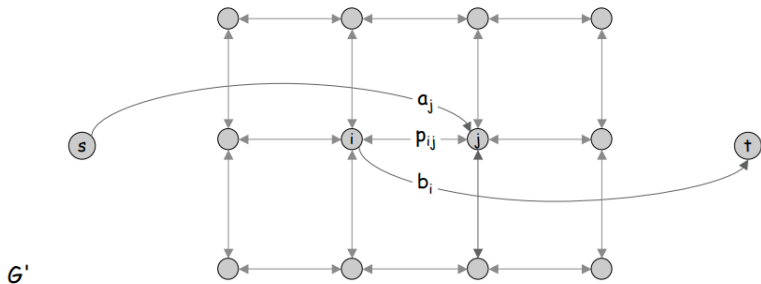
Die aus (S, T) abgeleitete Partition (A, B) von P hat Kosten

$$q'(A, B) = \text{cap}(S, T)$$

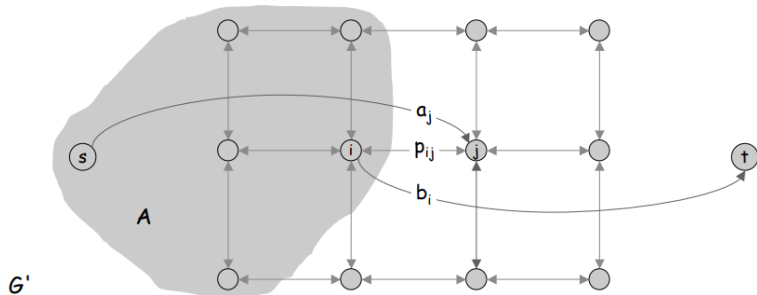
Schlussfolgerung: Eine optimale Segmentierung entspricht einem minimalen s - t -Schnitt im Netzwerk N .

- Kann mit einem Fluss- oder Schnittalgorithmus effizient bestimmt werden
- Praktisch: konkrete Gewichtungen $\alpha_p, \beta_p, \gamma_e$ hängen von Bildinhalten ab

Eine weitere Visualization- Bildsegmentierung



Eine weitere Visualization- Bildsegmentierung



Quellen - Links

- Image Segmentation with Code: <https://julie-jiang.github.io/image-segmentation/>
- Network Flow Applications: <https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pearson/07MaximumFlowApplications.pdf>

The End

