Strukturelle Modelle in der Bildverarbeitung
Binäre MinSum Probleme

D. Schlesinger – TUD/INF/KI/IS

- Äquivalente Transformationen (Reparametrisierung)
- Binäre MinSum Probleme – kanonische Form
- MinCut
- Binäre MinSum Probleme $\leftrightarrow$ MinCut
- Lösbarkeit, MinCut $\leftrightarrow$ MaxFlow
- MaxFlow Algorithmen

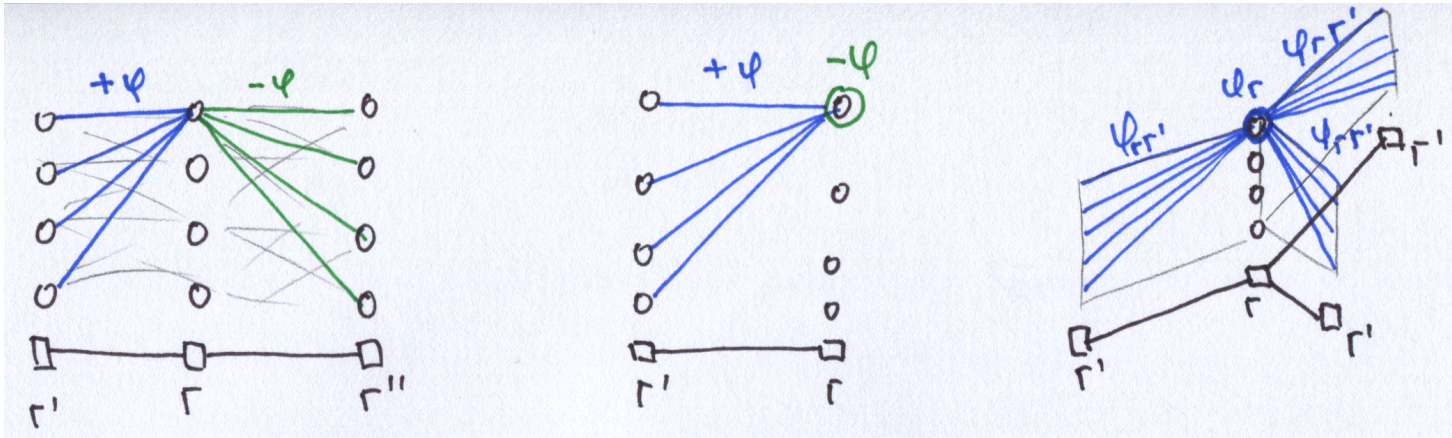
Äquivalente Transformationen (Reparametrisierung)

Zwei Aufgaben $A = (q, g)$ und $A' = (q', g')$ sind zu einander **äquivalent**, wenn

$$\left[\sum_r q_r(y_r) + \sum_{rr'} g_{rr'}(y_r, y_{r'})\right] = \left[\sum_r q'_r(y_r) + \sum_{rr'} g'_{rr'}(y_r, y_{r'})\right]$$

für alle Labellings y gilt.
$\mathcal{A}(A)$ – Äquivalenzklasse (alle zu A äquivalenten Aufgaben).

Äquivalente **Transformationen**:

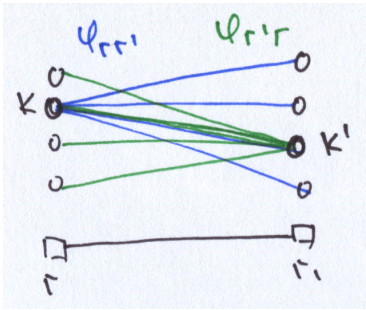


$$\Phi = \Big(\varphi_r(k)\ \forall r, k, \quad \varphi_{rr'}(k),\ \forall rr', k\Big)$$

$$\varphi_r(k) + \sum_{r':rr' \in E} \varphi_{rr'}(k) = 0 \quad \forall r, k$$

Äquivalente Transformationen

Sei $A = (q, g)$ eine Aufgabe, $A' = (q', g') = \Phi(A)$ ist die Aufgabe nach der Anwendung der Äquivalenten Transformation Φ, d.h.



$$q'_r(k) = q_r(k) + \varphi_r(k)$$
$$g'_{rr'}(k, k') = g_{rr'}(k, k') + \varphi_{rr'}(k) + \varphi_{r'r}(k')$$

$\Rightarrow$ A und A' sind zu einander äquivalent.

Sind zwei Aufgaben A und A' äquivalent,
so **existiert** eine Äquivalente Transformation Φ so, dass $A' = \Phi(A)$.

Weitere Eigenschaften:

$\Phi\Big(\Phi'(A)\Big) = \Phi'\Big(\Phi(A)\Big) = (\Phi \oplus \Phi')(A)$ – Superposition.

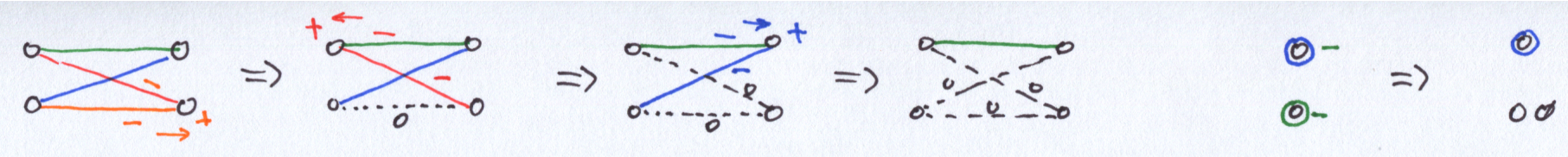
$\Phi^{-1}\Big(\Phi(A)\Big) = A$, d.h. $\Phi \oplus \Phi^{-1} = \Phi^0$ – Inverse Transformationen.

Die Menge aller Φ bildet eine **Gruppe**.

Zusätzliche „**konstante**“ Transformationen:
$q_r(k) += const, \forall k$ und $g_{rr'}(k, k') += const, \forall k, k'$
$\Rightarrow$ die Energien aller Labellings ändern sich um ein und dieselbe Konstante.

Binäre MinSum Probleme – kanonische Formen

Für **binäre** Probleme (d.h. $K = \{0, 1\}$) können die Funktionen q_r und $g_{rr'}$ mithilfe äquivalenter Transformationen wie folgt modifiziert werden:



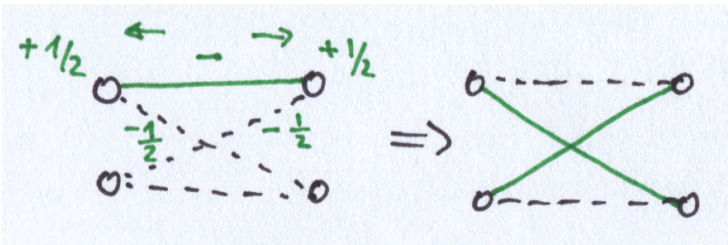
$\Rightarrow$ nur bei $k = 1$ und $(k, k') = (1, 1)$ sind die Werte von q_r bzw. $g_{rr'}$ ungleich 0.

$\Rightarrow$ die Energie lässt sich wie folgt schreiben:

$$E(y) = \sum_r y_r \cdot q_r + \sum_{rr'} y_r \cdot y_{r'} \cdot g_{rr'}$$

mit knoten- bzw. kantenspezifischen **Zahlen** q_r und $g_{rr'}$ – ein Polynom zweiter Ordnung – wird bei **Quadratic Pseudo-Boolean Optimization** (QPBO) verwendet.

Eine weitere Möglichkeit:

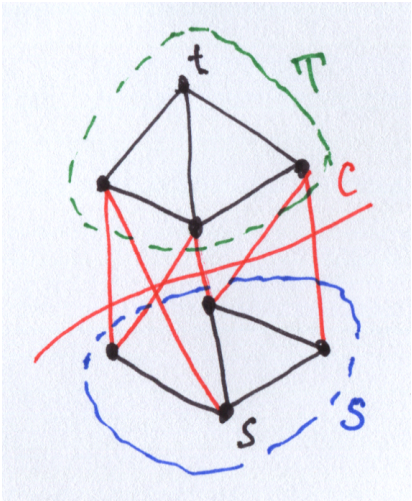


$$E(y) = (\ldots) + \sum_{rr'} g_{rr'} \cdot \mathbb{I}(y_r, y_{r'})$$

– wird bei der Überführung in **MinCut** verwendet.

MinCut

Achtung!!! Ähnliche Bezeichnungen, andere Bedeutung.



Gegeben sei in Graph $G = (R, E)$.

Es gibt zwei besondere Knoten – s (Source) und t (Target).

Jede Kante $\{r, r'\} \in E$ hat **Kosten** $c_{rr'}$.

Ein Schnitt (Cut) C ist eine Teilmenge der Kanten so, dass **kein Pfad** von s nach t existiert.

Zusätzlich ist der Schnitt im folgenden Sinne **minimal**: entfernt man aus C eine Kante, so ist er kein Schnitt mehr (ein Pfad von s nach t existiert).

Die Kosten eines Schnitts C sind die summarische Kosten seiner Kanten.

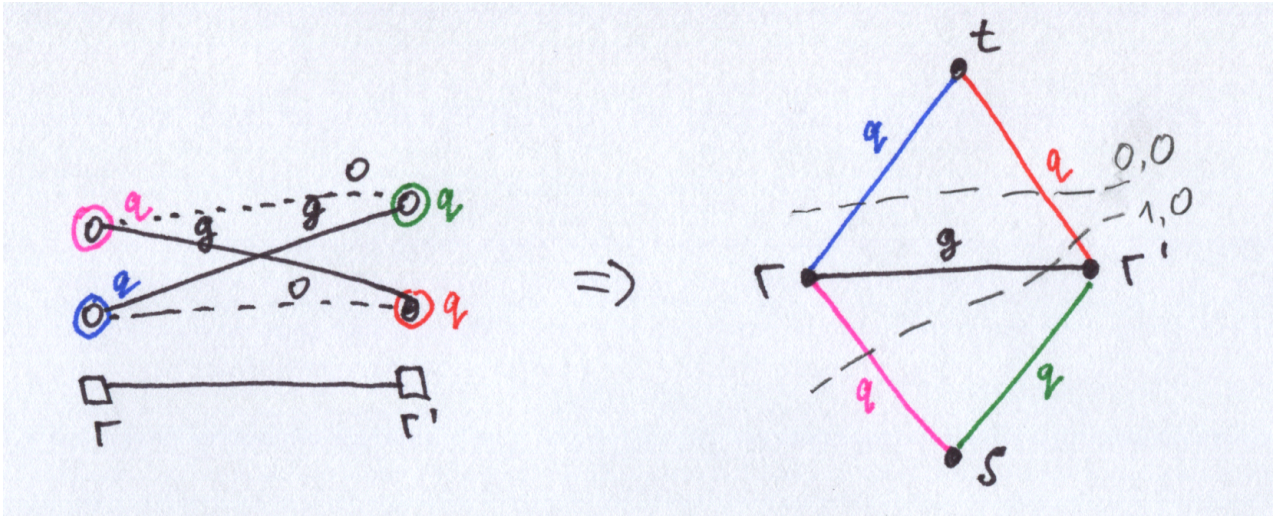
Gesucht wird der Schnitt minimaler Kosten:

$$C^* = \arg\min_{C} \sum_{rr' \in C} c_{rr'}$$

Alternativ: ein Schnitt entspricht einer **Partitionierung** der Menge der Knoten auf zwei Teilmengen S und T mit $s \in S$ und $t \in T$

$$(S, T)^* = \arg\min_{(S,T)} \sum_{rr' \in E, r \in S, t' \in T} c_{rr'}$$

Binäre MinSum Probleme ↔ MinCut



Jedem Knoten r des MinSum Problems entspricht ein „innerer“ Knoten r im MinCut.
Es gibt zwei zusätzliche Knoten s und t

Jedem Labelling $y : R \to \{0, 1\}$ entspricht eine Partitionierung (S, T), wobei
$y_r = 0 \Leftrightarrow r \in S$ und $y_r = 1 \Leftrightarrow r \in T$

Die Kantenkapazitäten des MinCut Problems sind:

$c_{rr'} = g_{rr'}$ für alle Kanten, die innere Knoten verbinden,

$c_{sr} = q_r(1)$ und $c_{rt} = q_r(0)$.

Die Energie eines Labellings y ist gleich den Kosten des entsprechenden Schnittes

$\Rightarrow$ Beste Partitionierung $(S, T)^*$ entspricht dem optimalen Labelling y^*

Lösbarkeit, MinCut $\leftrightarrow$ MaxFlow

Die Überführung MinSum $\leftrightarrow$ MinCut funktioniert **immer** (die Aufgaben sind identisch).
MinCut ist **NP** im Allgemeinen Fall.

Sind aber alle Kantenkosten eines MinCut Problems **nicht negativ**,
so ist das MinCut **polynomiell lösbar**.

MinCut Problem wird in das entsprechende **MaxFlow** Problem überführt.

Gegeben sei ein „Netz aus Rohren“ (ein Graph aus Knoten r und Kanten (r, r')).
Es gibt zwei besondere Knoten s und t.

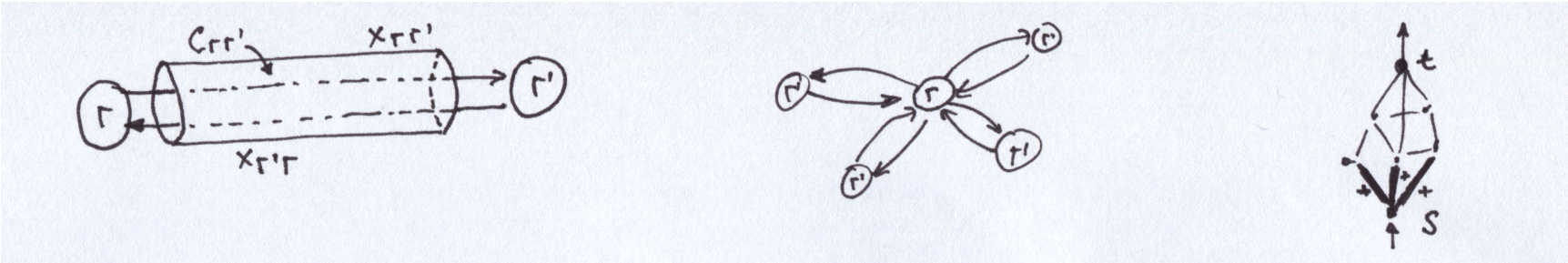
Durch jedes Rohr (r, r') fließt ein **Fluss** $x_{rr'}$

Jedes Rohr hat seine Kapazität $c_{rr'}$
– die maximale Menge des Flusses, die das Rohr durchlassen kann.

Gesucht wird nach der **maximalen** Menge des Flusses,
die durch das Netz von s nach t transportiert werden kann.

Sind die Kantenkosten eines MinCut Problems gleich den Kapazitäten der entsprechenden Rohren im MaxFlow Problem, so sind die Aufgaben zu einander **dual**.

Der Wert des maximalen Flusses ist gleich dem Wert des minimalen Schnittes.
Aus den optimalen Flüssen $x_{rr'}$ kann der optimale Schnitt ermittelt werden.



Flüsse sind **gerichtet**: $x_{rr'}$ bedeutet „von r nach r'“.

Flüsse sind durch Kapazitäten begrenzt:

$$0 \leq x_{rr'} \leq c_{rr'}$$

Nichts verschwindet oder entsteht unterwegs:

$$\sum_{r':rr' \in E} x_{r'r} = \sum_{r':rr' \in E} x_{rr'} \qquad \forall r \neq s, t$$

Maximiert wird das, was aus dem Knoten s ausfließest:

$$\sum_{r} x_{sr} \to \max_{x}$$

Algorithmus von Ford und Fulkerson

Restkapazitäten sind (gerichtete) Differenzen $r_{rr'} = c_{rr'} - x_{rr'}$
– wieviel Fluss diese Kante in dieser Richtung noch durchlassen kann.

Einen Fluss δ von r nach r' zu **schicken** heißt:

- die Restkapazität in dieser Richtung zu verkleinern: $r_{rr'} = r_{rr'} - \delta$
- die Restkapazität in umgekehrter Richtung zu vergrößern: $r_{r'r} = r_{r'r} + \delta$

Das geht nur, wenn δ kleiner als die aktuelle Restkapazität ist.
Ist ein Restkapazität null, so heißt die Kante in der entsprechenden Richtung **gesättigt**.

Algorithmus (Augmenting Path):

Initialisiere die Restkapazitäten mit $r_{rr'} = r_{r'r} = c_{rr'}$

Wiederhole:

1. Suche einen **noch nicht gesättigten Pfad** $(s, r_1, r_2 \dots t)$, d.h. $r_{r_i, r_{i+1}} > 0$ für alle i
   (das geht mit Dijkstra Algorithmus zur Suche nach dem besten Pfaden in Graphen)
2. Wenn es keinen gibt – Ende
3. Schicke Fluss $\delta = \min_i r_{r_i, r_{i+1}}$ entlang des Pfades
   (aktualisiere die Restkapazitäten dementsprechend), gehe zu 1.

Algorithmus von Ford und Fulkerson

Nach dem Anhalten des Algorithmus:

- Der Fluss ist maximal
- Die Kanten des optimalen Schnittes haben null Restkapazität
- umgekehrt gilt nicht
- Gibt es einen nicht gesättigter Pfad von s nach r, so $r \in S$
- Gibt es einen nicht gesättigter Pfad von r nach t, so $r \in T$

  (beides kann nicht sein, weil dann würde der Algorithmus noch nicht anhalten)

Interessant ist die Situation, wenn es Knoten r gibt,
für die weder ein Pfad von s nach r noch ein Pfad von r nach t existiert.
In dieser Situation gibt es **mehrere optimale** Schnitte.
Zwei davon ergeben sich, indem **alle** solche Knoten **einer** Teilmenge
(entweder alle zu S oder alle zu T) zuordnet werden (später noch etwas ausführlicher).

Es gibt weitere (bessere, unterschiedliche, für CV besser geeignete) Algorithmen

---

Zusammenfassung:

$$\text{Binäres MinSum Problem} \Leftrightarrow \text{Kanonische Form} \Leftrightarrow \text{MinCut} \Leftrightarrow \text{MaxFlow}$$