

# Maximale $s-t$ -Flüsse in Planaren Graphen

Vorlesung „Algorithmen für planare Graphen“ · 6. Juni 2017  
Guido Brückner

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER

# Maximales Flussproblem

Gegeben:

- Gerichteter Graph  $G = (V, E)$ ,  $\text{rev}(u \rightarrow v) := (v \rightarrow u)$
- $c: E \rightarrow \mathbb{R}$ ,  $(\text{OE } \text{rev}(e) \in E \quad \forall e \in E)$
- $s, t \in V$

Gesucht:

Zulässiger Fluss  $\phi$  mit maximalem  $\sum_{s \rightarrow v \in E} \phi(s \rightarrow v)$ .

# Maximales Flussproblem

Gegeben:

- Gerichteter Graph  $G = (V, E)$ ,  $\text{rev}(u \rightarrow v) := (v \rightarrow u)$
- $c: E \rightarrow \mathbb{R}$ ,  $(\text{OE } \text{rev}(e) \in E \quad \forall e \in E)$
- $s, t \in V$

Gesucht:

Zulässiger Fluss  $\phi$  mit maximalem  $\sum_{s \rightarrow v \in E} \phi(s \rightarrow v)$ .

$\phi: E \rightarrow \mathbb{R}$  ***st-Fluß***:

- $\phi(e) = -\phi(\text{rev}(e))$  **(Symmetrie)**
- $\phi(e) \leq c(e)$  **(Zulässigkeit)**
- $\forall v \in V \setminus \{s, t\} : \sum_w \phi(v \rightarrow w) = 0$  **(Flusserhaltung)**

Gegeben:

- Gerichteter Graph  $G = (V, E)$ ,  $\text{rev}(u \rightarrow v) := (v \rightarrow u)$
- $c: E \rightarrow \mathbb{R}$ ,  $(\text{OE } \text{rev}(e) \in E \quad \forall e \in E)$
- $s, t \in V$

Gesucht:

Zulässiger Fluss  $\phi$  mit maximalem  $\sum_{s \rightarrow v \in E} \phi(s \rightarrow v)$ .

$\exists$  viele verschiedene Lösungsalgorithmen, deren Laufzeit aber mindestens  $\mathcal{O}(n^2)$  ist.

**Heute:** ein Algorithmus für planare Graphen mit  $\mathcal{O}(n \log n)$  Laufzeit.

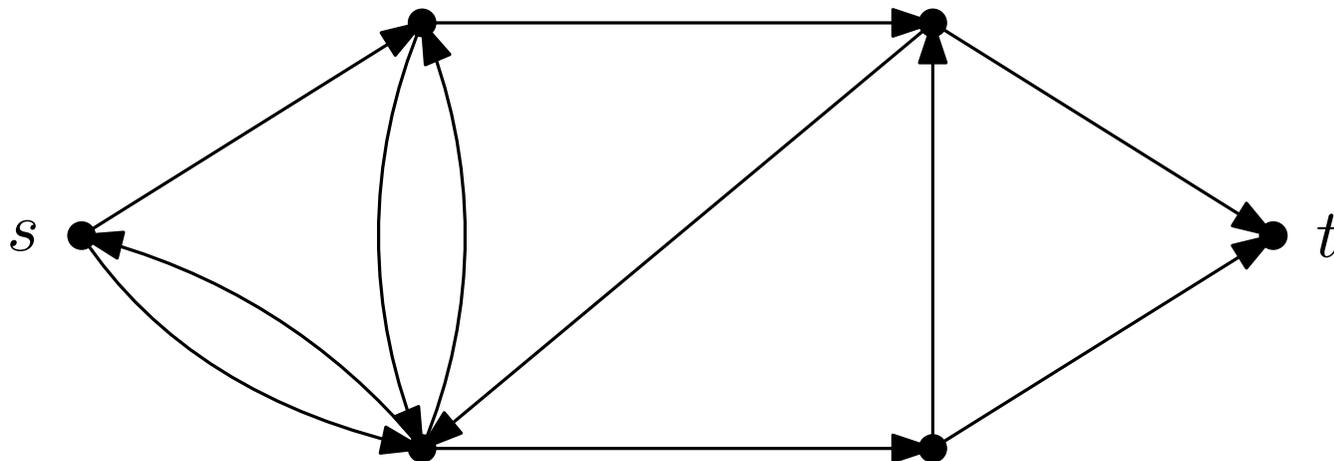
- **Teil I:** maximaler Fluss in  $st$ -planaren Graphen
- **Teil II:** maximaler Fluss in allgemeinen planaren Graphen

# Maximaler Fluss in $st$ -planaren Graphen

Graph  $G = (V, E)$  mit  $s, t \in V$  ist  **$st$ -planar**  $\iff$   $G$  lässt sich so einbetten, dass  $s, t$  inzident zur äußeren Facette

# Maximaler Fluss in $st$ -planaren Graphen

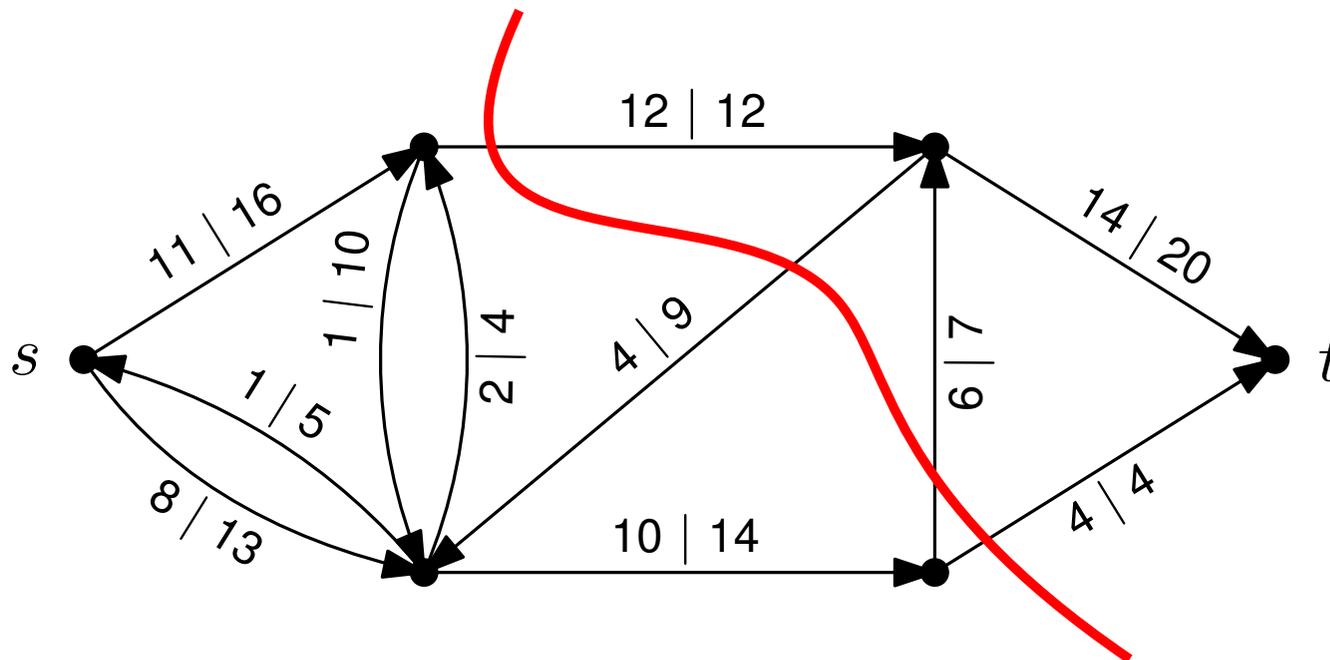
Graph  $G = (V, E)$  mit  $s, t \in V$  ist  **$st$ -planar**  $\iff G$  lässt sich so einbetten, dass  $s, t$  inzident zur äußeren Facette



## Lemma

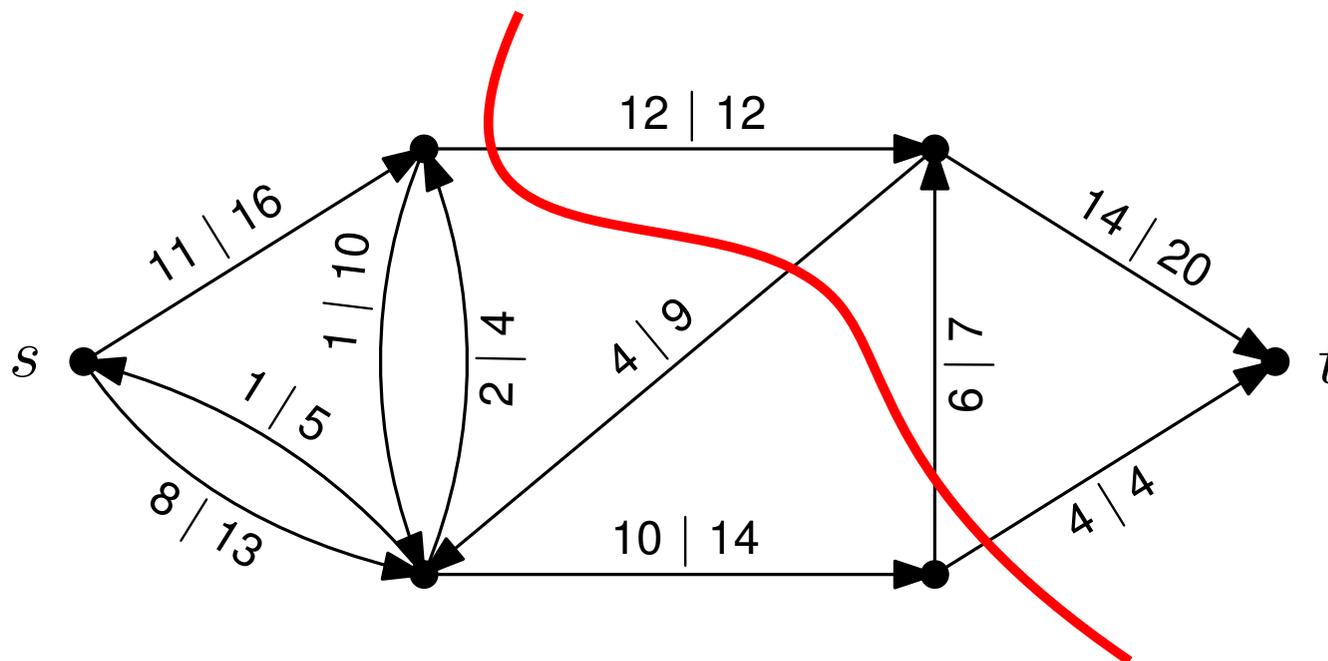
Sei  $\phi$  ein  $st$ -Fluss und  $E(S, V \setminus S)$  ein  $st$ -Schnitt. Dann gilt:

$$w(\phi) = \sum_{e \in E(S, V \setminus S)} \phi(e) - \sum_{e \in E(V \setminus S, S)} \phi(e) \leq c(S, V \setminus S)$$



## Korollar

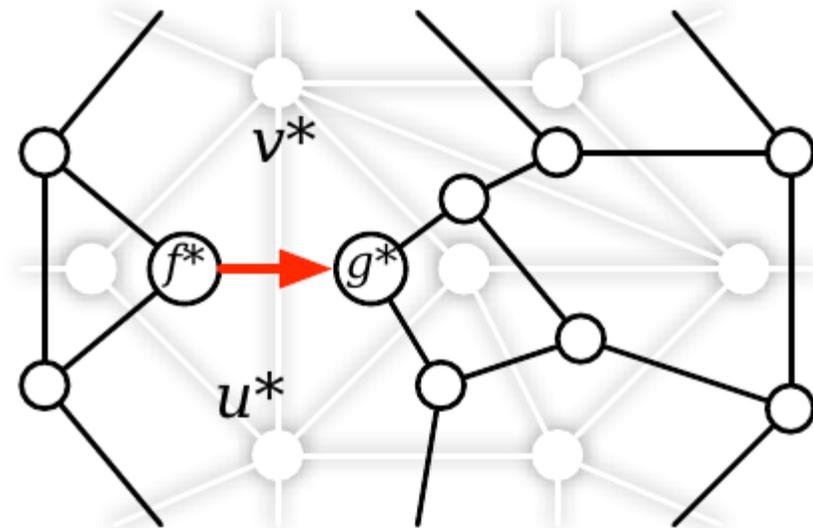
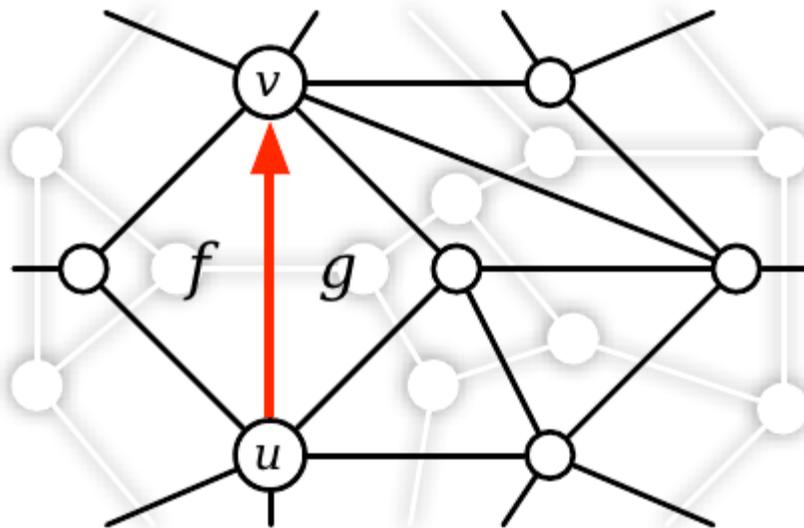
Ein  $st$ -Fluss  $\phi$  mit  $w(\phi) = c(S, V \setminus S)$  ist maximal.



# Dual-Graphen, gerichtet(!)

$G = (V, E)$  gerichtet

**Dualgraph:** Wie gehabt,  $(u \rightarrow v)^*$  kreuzt  $u \rightarrow v$  von links nach rechts!

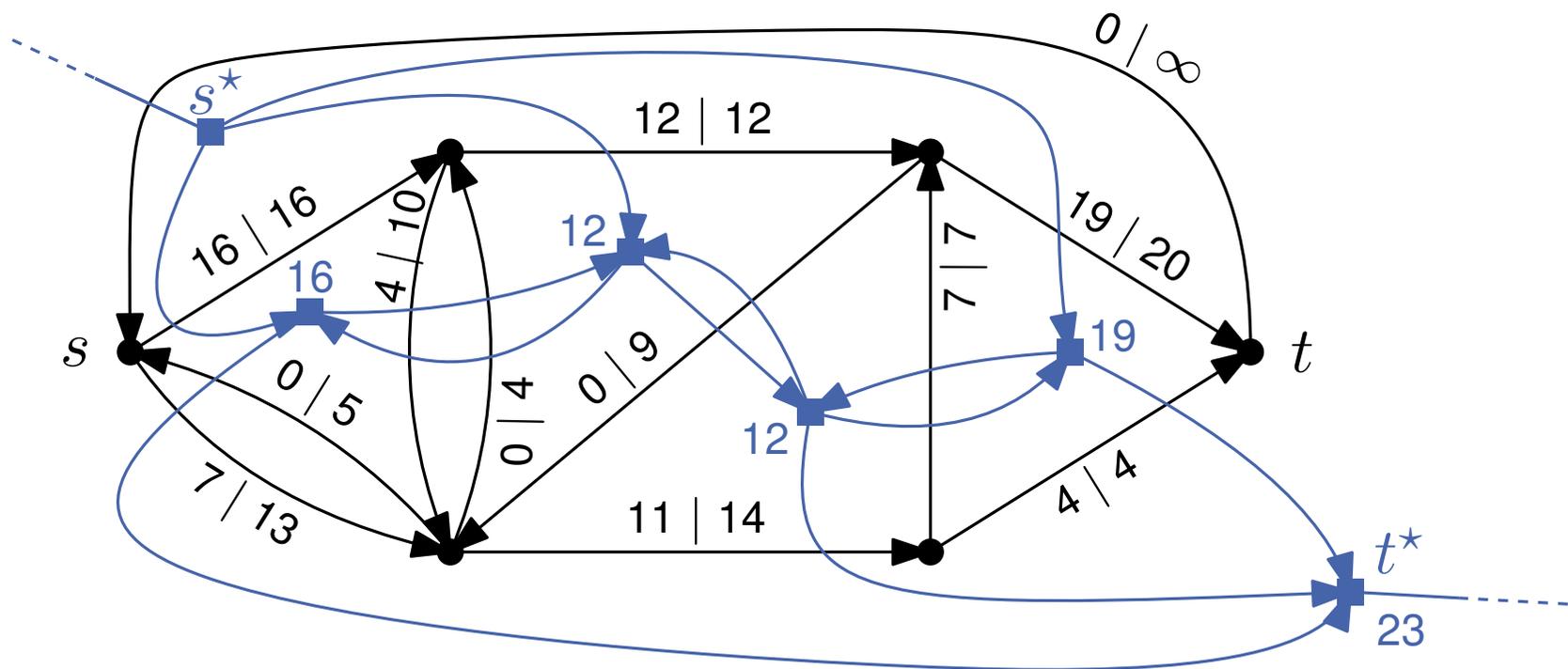


[Erickson'10]

# Maximaler Fluss in $st$ -planaren Graphen

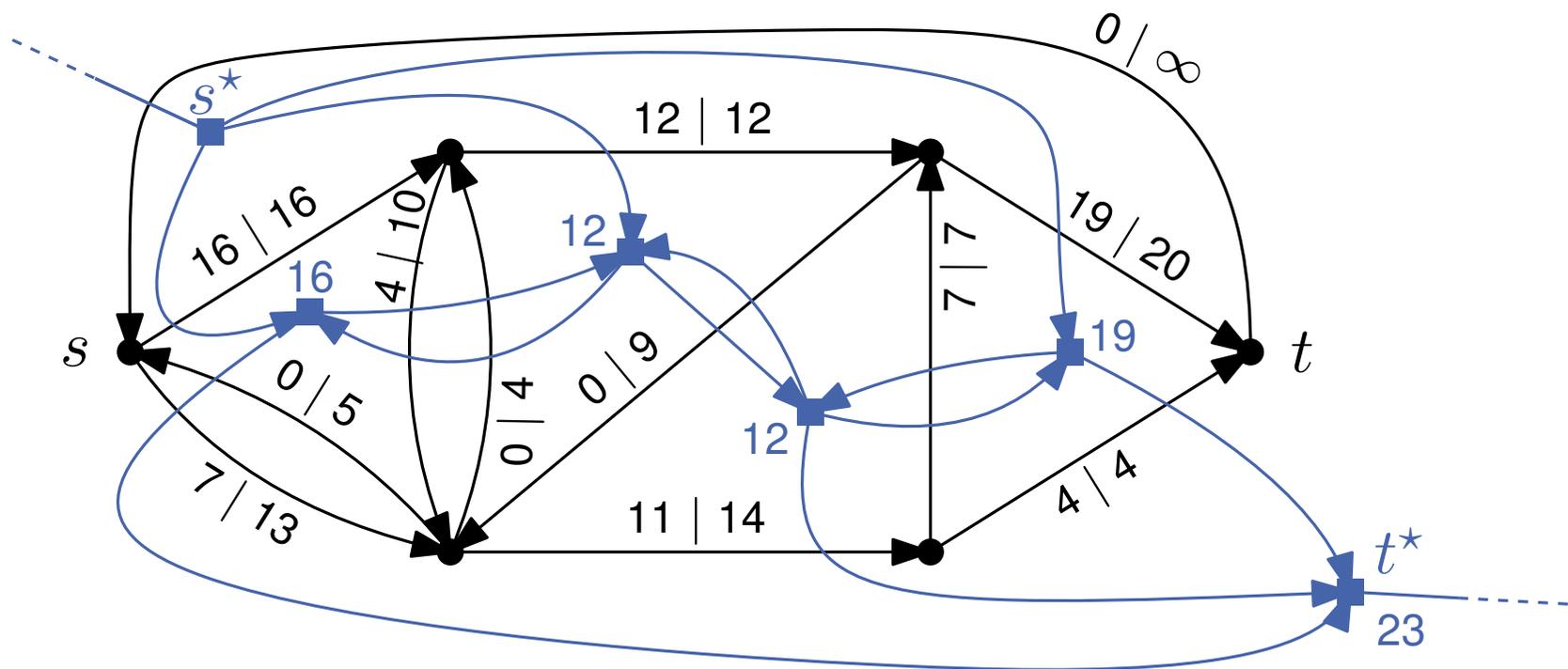
## Idee:

- füge Rückkante  $t \rightarrow s$  ein
- konstruiere gerichteten Dualgraph  $G^*$ , setze  $\ell(e^*) := c(e)$
- suche kürzesten  $s^*-t^*$ -Pfad



## Lemma

Die Funktion  $\phi(e) := \max(0, d(s^*, \text{right}(e)) - d(s^*, \text{left}(e)))$  beschreibt einen maximalen  $st$ -Fluss.



## Algorithmus:

- konstruiere Dualgraph  $\mathcal{O}(n)$
- finde kürzesten  $st$ -Pfad
  - z.B. mit Dijkstra  $\mathcal{O}(n \log n)$
  - aber für planare Graphen schneller  $\mathcal{O}(n)$   
(Henzinger et al., 1997)

## Theorem

Für  $st$ -planare Graphen lässt sich ein maximaler Fluss in Linearzeit berechnen.

- **Teil I:** maximaler Fluss in  $st$ -planaren Graphen
- **Teil II:** maximaler Fluss in allgemeinen planaren Graphen

## Parametrisches Kürzeste-Wege-Problem:

Gegeben: Gerichteter Graph  $G = (V, E)$ ,  $E' \subseteq E$ ,  $c: E \rightarrow \mathbb{R}$

$c(\lambda, e) = c(e) - \lambda$  für  $e \in E'$ ,  $c(\lambda, e) = c(e)$  für  $e \notin E'$

Gesucht: Größtes  $\lambda$  mit  $G$  enthält bzgl.  $c(\lambda, \cdot)$  keine negativen Kreise.

Lösbar in Zeit  $O(nm \log n)$  [Karp, Orlin] bzw.  $O(n^2 \log n)$  [Young et al.]

## Parametrisches Kürzeste-Wege-Problem:

Gegeben: Gerichteter Graph  $G = (V, E)$ ,  $E' \subseteq E$ ,  $c: E \rightarrow \mathbb{R}$

$c(\lambda, e) = c(e) - \lambda$  für  $e \in E'$ ,  $c(\lambda, e) = c(e)$  für  $e \notin E'$

Gesucht: Größtes  $\lambda$  mit  $G$  enthält bzgl.  $c(\lambda, \cdot)$  keine negativen Kreise.

Lösbar in Zeit  $O(nm \log n)$  [Karp, Orlin] bzw.  $O(n^2 \log n)$  [Young et al.]

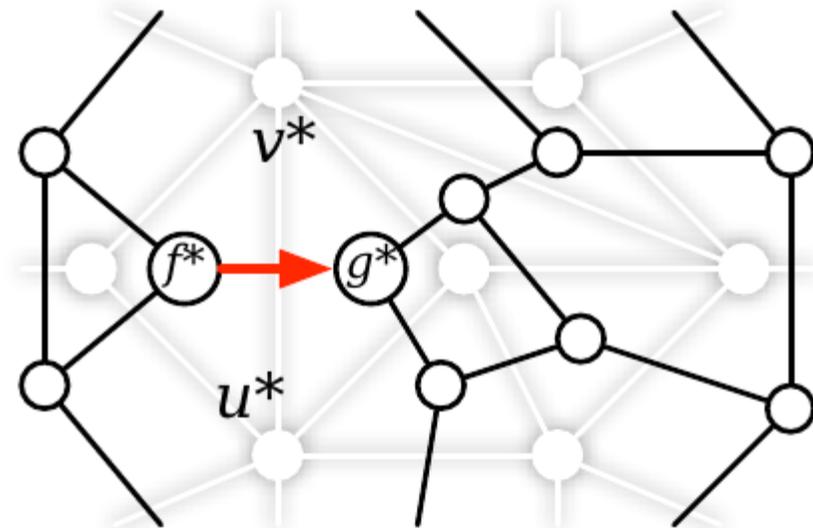
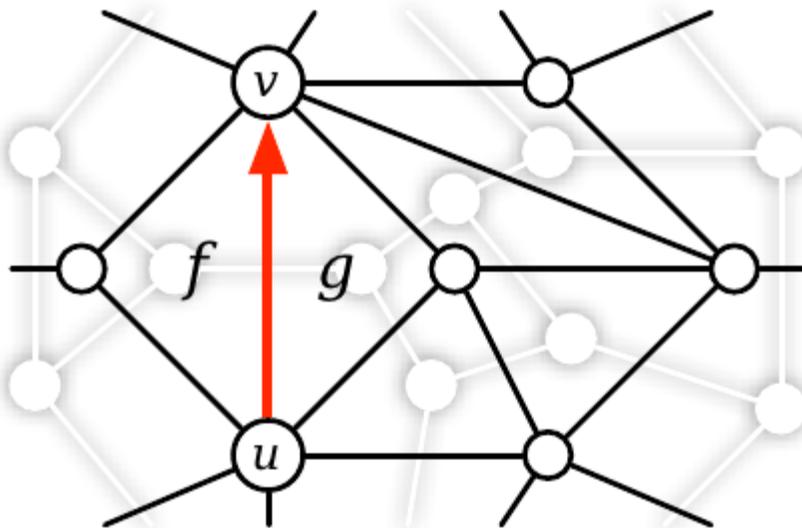
Idee:

- Betrachte kürzeste-Wege-Baum bzgl.  $c(\lambda, \cdot)$  bei bel. Startknoten  $s$ .
- Erhöhe  $\lambda$  schrittweise.
- $\text{dist}(\lambda, v) \leq \text{dist}(\lambda, u) + c(\lambda, u \rightarrow v)$
- Kritische Stelle bei Gleichheit,  
 $u \rightarrow v$  kommt in den Baum, ersetzt  $u' \rightarrow v$ .
- Jeder Pivot-Schritt erhöht Anzahl Kanten in  $E'$  auf einem  $s-v$ -Pfad  
 $\rightsquigarrow O(n^2)$  Pivot-Schritte

# Dual-Graphen, gerichtet(!)

$G = (V, E)$  gerichtet

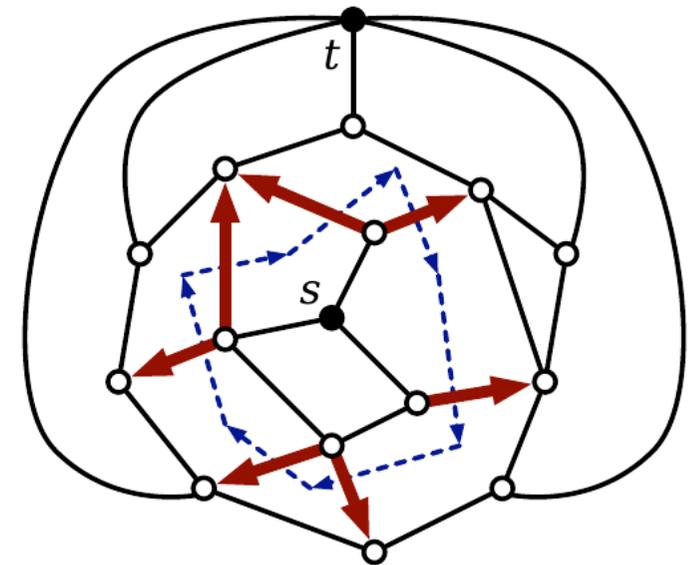
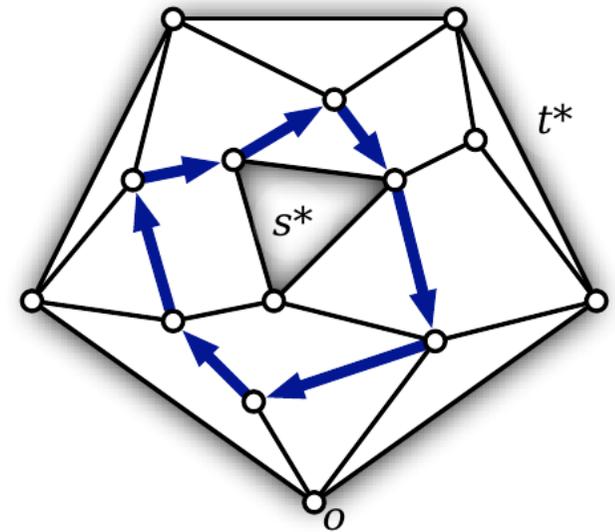
**Dualgraph:** Wie gehabt,  $(u \rightarrow v)^*$  kreuzt  $u \rightarrow v$  von links nach rechts!



[Erickson'10]

## Gerichteter $(s, t)$ -Schnitt:

Kantenmenge  $C$ , sodass jeder gerichtete  $st$ -Pfad Kante aus  $C$  enthält.



## Gerichteter $(s, t)$ -Schnitt:

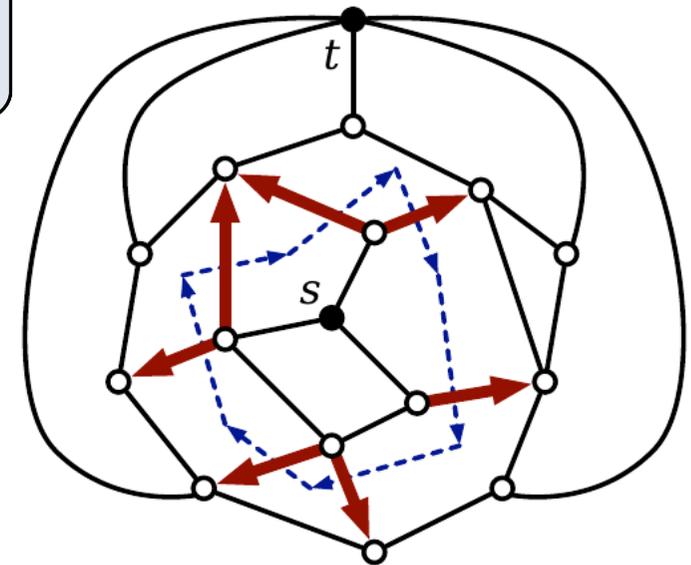
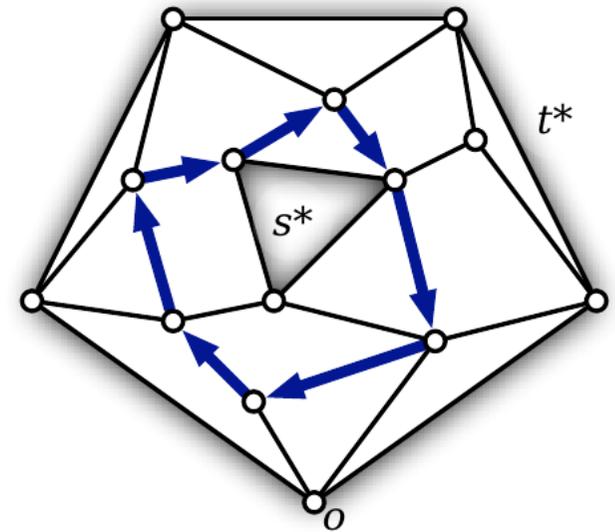
Kantenmenge  $C$ , sodass jeder gerichtete  $st$ -Pfad Kante aus  $C$  enthält.

## Dualität:

Gerichteter  $(s, t)$ -Schnitt in  $G$

$\equiv$

gerichteter Kreis in  $G^*$ , der  $s^*$  und  $t^*$  trennt.



## Gerichteter $(s, t)$ -Schnitt:

Kantenmenge  $C$ , sodass jeder gerichtete  $st$ -Pfad Kante aus  $C$  enthält.

## Dualität:

Gerichteter  $(s, t)$ -Schnitt in  $G$

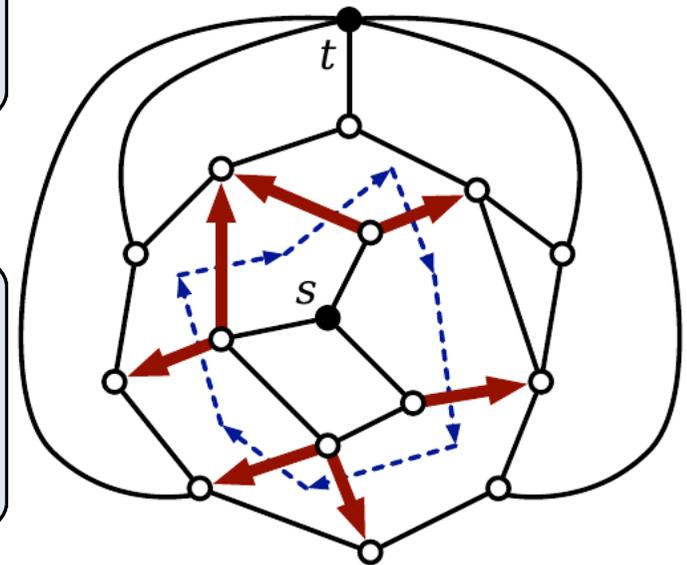
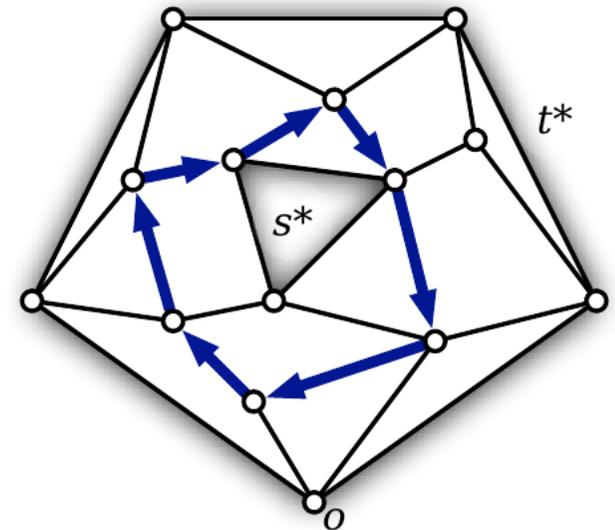
$\equiv$

gerichteter Kreis in  $G^*$ , der  $s^*$  und  $t^*$  trennt.

Kantenmenge  $C \subseteq E$  **Kozykel**

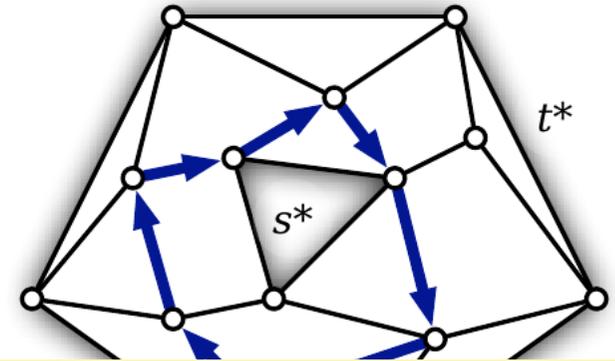
$:\Leftrightarrow$

$C^*$  einfacher gerichteter Kreis



## Gerichteter $(s, t)$ -Schnitt:

Kantenmenge  $C$ , sodass jeder gerichtete  $st$ -Pfad Kante aus  $C$  enthält.



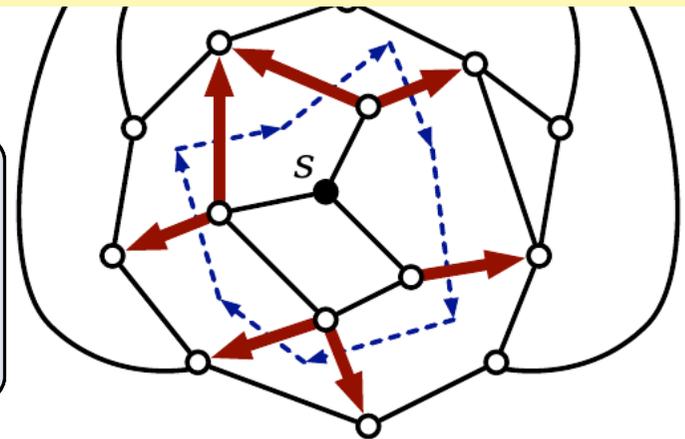
**Idee:** ähnlich wie bei  $st$ -planaren Graphen, aber *kürzeste Kreise* statt *kürzester Pfade*

**Aber:** wie kürzeste Kreise effizient finden?

Kantenmenge  $C \subseteq E$  **Kozykel**

$:\Leftrightarrow$

$C^*$  einfacher gerichteter Kreis



# Kozykel und st-Schnitte

$P$  beliebiger gerichteter Pfad von  $s$  nach  $t$ .

Definiere  $\pi : E \rightarrow \mathbb{R}$ , **Einheitsfluß** auf  $P$

$$\pi(e) := \begin{cases} 1 & e \in P \\ -1 & \text{rev}(e) \in P \\ 0 & \text{sonst} \end{cases}$$

Für  $E' \subseteq E$  definiere:  $\pi(E') := \sum_{e \in E'} \pi(e)$

# Kozykel und st-Schnitte

$P$  beliebiger gerichteter Pfad von  $s$  nach  $t$ .

Definiere  $\pi: E \rightarrow \mathbb{R}$ , **Einheitsfluß** auf  $P$

$$\pi(e) := \begin{cases} 1 & e \in P \\ -1 & \text{rev}(e) \in P \\ 0 & \text{sonst} \end{cases}$$

Für  $E' \subseteq E$  definiere:  $\pi(E') := \sum_{e \in E'} \pi(e)$

## Lemma

Für jeden Kozykel  $C$  gilt:  $\pi(C) \in \{-1, 0, 1\}$

$$\pi(C) = 1 \quad \Leftrightarrow \quad C \text{ ist } (s, t)\text{-Schnitt.}$$

$P$  beliebiger gerichteter Pfad von  $s$  nach  $t$ .

Definiere  $\lambda \cdot \pi : E \rightarrow \mathbb{R}$ ,  **$\lambda$ -Fluss** auf  $P$

$$\lambda \cdot \pi(e) := \begin{cases} \lambda & e \in P \\ -\lambda & \text{rev}(e) \in P \\ 0 & \text{sonst} \end{cases}$$

**Vorgehen:** Erhöhe  $\lambda$  schrittweise. Was passiert dann mit Kreisen?

- Kreise, die  $s$  enthalten, werden um  $\lambda$  kürzer
- Kreise, die  $t$  enthalten, werden um  $\lambda$  länger
- Kreise, die  $s$  und  $t$  nicht trennen, bleiben gleich lang

**Vorgehen:** Erhöhe  $\lambda$  schrittweise. Was passiert dann mit Kreisen?

- Kreise, die  $s$  enthalten, werden um  $\lambda$  kürzer
- Kreise, die  $t$  enthalten, werden um  $\lambda$  länger
- Kreise, die  $s$  und  $t$  nicht trennen, bleiben gleich lang

**Also:** Erhöhe  $\lambda$  solange, bis ein Kreis der Länge 0 entsteht. Das kann man effizient feststellen.

# Betrachte Fluß von $\lambda$ auf $P$

Setze  $\phi := \lambda \cdot \pi$

Betrachte **Residual-Netzwerk**  $G_\lambda := G_{\lambda \cdot \phi}$

Das ist  $G$  mit residualer Kapazitätsfunktion  $c(\lambda, e) := c(e) - \lambda \cdot \pi(e)$

$G_\lambda^*$ : Dual-Graph mit  $e^*$  hat Kosten  $c(\lambda, e^*) := c(\lambda, e)$ .

# Betrachte Fluß von $\lambda$ auf $P$

Setze  $\phi := \lambda \cdot \pi$

Betrachte **Residual-Netzwerk**  $G_\lambda := G_{\lambda \cdot \phi}$

Das ist  $G$  mit residualer Kapazitätsfunktion  $c(\lambda, e) := c(e) - \lambda \cdot \pi(e)$

$G_\lambda^*$ : Dual-Graph mit  $e^*$  hat Kosten  $c(\lambda, e^*) := c(\lambda, e)$ .

## Lemma

$G$  besitzt gültigen  $s - t$ -Fluß mit Wert  $\lambda$

$\Leftrightarrow$

$G_\lambda^*$  enthält keinen negativen Kreis

# Betrachte Fluß von $\lambda$ auf $P$

Setze  $\phi := \lambda \cdot \pi$

Betrachte **Residual-Netzwerk**  $G_\lambda := G_{\lambda \cdot \phi}$

Das ist  $G$  mit residualer Kapazitätsfunktion  $c(\lambda, e) := c(e) - \lambda \cdot \pi(e)$

$G_\lambda^*$ : Dual-Graph mit  $e^*$  hat Kosten  $c(\lambda, e^*) := c(\lambda, e)$ .

## Lemma

$G$  besitzt gültigen  $s - t$ -Fluß mit Wert  $\lambda$

$\Leftrightarrow$

$G_\lambda^*$  enthält keinen negativen Kreis

$\rightsquigarrow$  parametrisches Kürzeste-Wege-Problem mit Koeffizienten  $-1, 0, 1$

## Lemma

$G$  besitzt gültigen  $s - t$ -Fluß mit Wert  $\lambda$



$G_{\lambda}^*$  enthält keinen negativen Kreis

Löse also spezielles parametrisches Kürzeste-Wege-Problem  
Koeffizienten  $-1, 0, 1$

## Lemma

$G$  besitzt gültigen  $s - t$ -Fluß mit Wert  $\lambda$

$\Leftrightarrow$

$G_\lambda^*$  enthält keinen negativen Kreis

Löse also spezielles parametrisches Kürzeste-Wege-Problem  
Koeffizienten  $-1, 0, 1$

Verwalte Kürzeste-Wege-Baum  $T_\lambda$  in  $G_\lambda^*$  mit Wurzel  $o$ .

PlanarMaxFlow( $G, c, s, t$ )

- Berechnet  $T_0$
- Verwalte  $T_\lambda$ , während  $\lambda$  kontinuierlich von 0 bis  $\lambda_{\max}$  läuft.
- Berechne  $\phi(\lambda_{\max}, \cdot)$  aus  $T_{\lambda_{\max}}$ .

# Abbruchkriterium

Speichere Baum mittels Vorgängerfunktion  $\text{pred}(p, \lambda)$ .

Duale Kante  $e^*$  **straff** bei  $\lambda \iff \text{slack}(\lambda, e^*) = 0$ .

# Abbruchkriterium

Speichere Baum mittels Vorgängerfunktion  $\text{pred}(p, \lambda)$ .

Duale Kante  $e^*$  **straff** bei  $\lambda \iff \text{slack}(\lambda, e^*) = 0$ .

Außer bei kritischen  $\lambda$ -Werten:  $e^*$  straff  $\iff e^*$  in  $T_\lambda$

Bei kritischen Werten:

Duale Nichtbaumkante  $p \rightarrow q$  wird straff.

$\rightsquigarrow p \rightarrow q$  ersetzt  $\text{pred}(\lambda, q) \rightarrow q$

# Abbruchkriterium

Speichere Baum mittels Vorgängerfunktion  $\text{pred}(p, \lambda)$ .

Duale Kante  $e^*$  **straff** bei  $\lambda \iff \text{slack}(\lambda, e^*) = 0$ .

Außer bei kritischen  $\lambda$ -Werten:  $e^*$  straff  $\iff e^*$  in  $T_\lambda$

Bei kritischen Werten:

Duale Nichtbaumkante  $p \rightarrow q$  wird straff.

$\rightsquigarrow p \rightarrow q$  ersetzt  $\text{pred}(\lambda, q) \rightarrow q$

**Pivotschritt**

# Abbruchkriterium

Speichere Baum mittels Vorgängerfunktion  $\text{pred}(p, \lambda)$ .

Duale Kante  $e^*$  **straff** bei  $\lambda \iff \text{slack}(\lambda, e^*) = 0$ .

Außer bei kritischen  $\lambda$ -Werten:  $e^*$  straff  $\iff e^*$  in  $T_\lambda$

Bei kritischen Werten:

Duale Nichtbaumkante  $p \rightarrow q$  wird straff.

**Pivotschritt**

$\rightsquigarrow p \rightarrow q$  ersetzt  $\text{pred}(\lambda, q) \rightarrow q$

## Lemma

$\lambda_{\max}$  ist erster kritischer Wert von  $\lambda$ , dessen Pivot einen gerichteten Kreis in  $T_\lambda$  erzeugt.

# Abbruchkriterium

Speichere Baum mittels Vorgängerfunktion  $\text{pred}(p, \lambda)$ .

Duale Kante  $e^*$  **straff** bei  $\lambda \iff \text{slack}(\lambda, e^*) = 0$ .

Außer bei kritischen  $\lambda$ -Werten:  $e^*$  straff  $\iff e^*$  in  $T_\lambda$

Bei kritischen Werten:

Duale Nichtbaumkante  $p \rightarrow q$  wird straff.

**Pivotschritt**

$\rightsquigarrow p \rightarrow q$  ersetzt  $\text{pred}(\lambda, q) \rightarrow q$

## Lemma

$\lambda_{\max}$  ist erster kritischer Wert von  $\lambda$ , dessen Pivot einen gerichteten Kreis in  $T_\lambda$  erzeugt.

Primale Kante  $e$  ist **locker** wenn weder  $e^*$  noch  $\text{rev}(e^*)$  straff.

$L_\lambda$ : lockere Kanten

Außer bei kritischen Werten:  $L_\lambda$  Spannbaum von  $G$ .

# Abbruchkriterium

Speichere Baum mittels Vorgängerfunktion  $\text{pred}(p, \lambda)$ .

Duale Kante  $e^*$  **straff** bei  $\lambda \iff \text{slack}(\lambda, e^*) = 0$ .

Außer bei kritischen  $\lambda$ -Werten:  $e^*$  straff  $\iff e^*$  in  $T_\lambda$

Bei kritischen Werten:

Duale Nichtbaumkante  $p \rightarrow q$  wird straff.

**Pivotschritt**

$\rightsquigarrow p \rightarrow q$  ersetzt  $\text{pred}(\lambda, q) \rightarrow q$

## Lemma

$\lambda_{\max}$  ist erster kritischer Wert von  $\lambda$ , dessen Pivot einen gerichteten Kreis in  $T_\lambda$  erzeugt.

Primale Kante  $e$  ist **locker** wenn weder  $e^*$  noch  $\text{rev}(e^*)$  straff.

$L_\lambda$ : lockere Kanten

Außer bei kritischen Werten:  $L_\lambda$  Spannbaum von  $G$ .

## Lemma

$\lambda_{\max}$  ist erster kritischer Wert von  $\lambda$ , dessen Pivot  $L_\lambda$  unzusammenhängend macht.

Duale Kante **aktiv** wenn slack mit  $\lambda$  abnimmt.

$L_\lambda$  enthält eindeutigen  $st$ -Pfad  $LP_\lambda$ .

### Lemma

Duale Kante  $e^*$  aktiv  $\Leftrightarrow e$  Kante von  $LP_\lambda$ .

PlanarMaxFlow( $G, c, s, t$ )

Initialisiere Spannbaum  $L$ , Vorgänger und Schlupfvariablen.

**while**  $s, t$  in selber Komponente von  $L$  **do**

$LP \leftarrow$  st-Pfad in  $L$

$p \rightarrow q \leftarrow$  Kante in  $LP^*$  mit minimalem Schlupf

$\delta \leftarrow$  slack( $p \rightarrow q$ )

**for** jede Kante  $e$  in  $LP$  **do**

        | slack( $e^*$ )  $\leftarrow$  slack( $e^*$ )  $- \delta$

        | slack(rev( $e^*$ ))  $\leftarrow$  slack(rev( $e^*$ ))  $+ \delta$

**end**

    Entferne  $(p \rightarrow q)^*$  aus  $L$

**if**  $q \neq o$  **then**

        | Füge  $(\text{pred}(q) \rightarrow q)^*$  in  $L$  ein

**end**

    pred( $q$ )  $\leftarrow p$

**end**

**for** jede Kante  $e$  **do**

    |  $\blacksquare(e) \leftarrow c(e) - \text{slack}(e^*)$

**end**

PlanarMaxFlow( $G, c, s, t$ )

Initialisiere Spannbaum  $L$ , Vorgänger und Schlupfvariablen.

**while**  $s, t$  in selber Komponente von  $L$  **do**

$LP \leftarrow$  st-Pfad in  $L$

$p \rightarrow q \leftarrow$  Kante in  $LP^*$  mit minimalem Schlupf

Datenstruktur Top-Tree verwaltet dynamischen Wald  $L_\lambda$ .

Erlaubt folgende Operationen mit  $O(\log n)$  Zeit:

- Löschen und Einfügen von Kanten
- Anfragen ob zwei Knoten in selber Zusammenhangskomponente
- Finden des kleinsten Gewichts auf einem Pfad
- Modifizieren aller Gewichte auf einem Pfad um denselben Wert

**end**

$\text{pred}(q) \leftarrow p$

**end**

**for jede Kante**  $e$  **do**

$w(e) \leftarrow c(e) - \text{slack}(e^*)$

**end**

PlanarMaxFlow( $G, c, s, t$ )

Initialisiere Spannbaum  $L$ , Vorgänger und Schlupfvariablen.

**while**  $s, t$  in selber Komponente von  $L$  **do**  $O(\log n)$

$LP \leftarrow$  st-Pfad in  $L$

$p \rightarrow q \leftarrow$  Kante in  $LP^*$  mit minimalem Schlupf  $O(\log n)$

$\delta \leftarrow$  slack( $p \rightarrow q$ )

**for** jede Kante  $e$  in  $LP$  **do**

        | slack( $e^*$ )  $\leftarrow$  slack( $e^*$ )  $- \delta$   $O(\log n)$

        | slack(rev( $e^*$ ))  $\leftarrow$  slack(rev( $e^*$ ))  $+ \delta$

**end**

    Entferne  $(p \rightarrow q)^*$  aus  $L$   $O(\log n)$

**if**  $q \neq o$  **then**

        | Füge  $(\text{pred}(q) \rightarrow q)^*$  in  $L$  ein  $O(\log n)$

**end**

$\text{pred}(q) \leftarrow p$

**end**

**for** jede Kante  $e$  **do**

    |  $\blacksquare(e) \leftarrow c(e) - \text{slack}(e^*)$   $O(n)$

**end**

- Iteration lässt sich in  $O(\log n)$  Zeit implementieren mittels top-trees
  - Spezielle Struktur  $\Rightarrow O(n)$  Pivot-Schritte, (ohne Beweis)  
d.h.  $O(n)$  Durchläufe der while-Schleife
- $\rightsquigarrow$  Gesamtlaufzeit  $O(n \log n)$

- Iteration lässt sich in  $O(\log n)$  Zeit implementieren mittels top-trees
  - Spezielle Struktur  $\Rightarrow O(n)$  Pivot-Schritte, (ohne Beweis)  
d.h.  $O(n)$  Durchläufe der while-Schleife
- $\rightsquigarrow$  Gesamtlaufzeit  $O(n \log n)$

siehe

”Maximum Flows and Parametric Shortest Paths in Planar Graphs”,  
Jeff Erickson, SODA’10

Algorithmus kann als iterative Augmentierung entlang links-liegendster Pfade aufgefasst werden.