

Maximale $s-t$ -Flüsse in Planaren Graphen

Vorlesung "Algorithmen für planare Graphen" · June 1, 2015
Ignaz Rutter

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

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

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

- $\phi(e) = -\phi(\text{rev}(e))$
- $\sum_w \phi(v \rightarrow w) = 0$ für alle $v \in V \setminus \{s, t\}$

ϕ **zulässig**: $\phi(e) \leq c(e)$ für alle $e \in E$

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 λ 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 λ 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.]

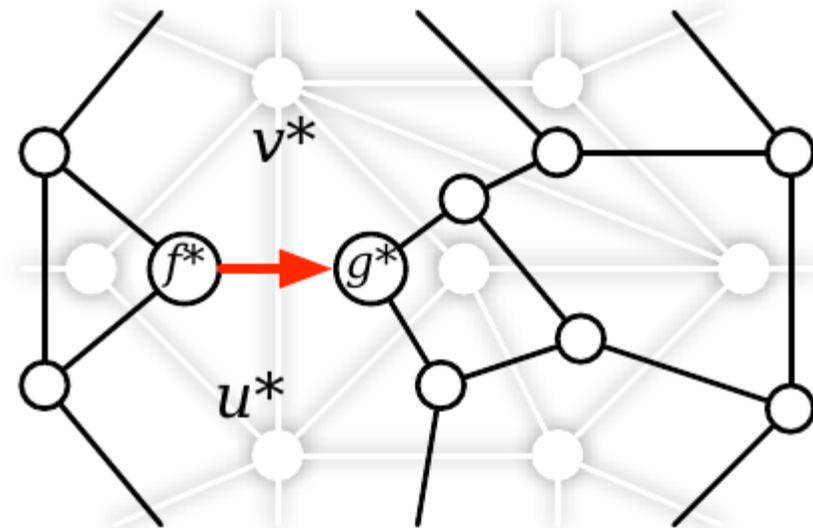
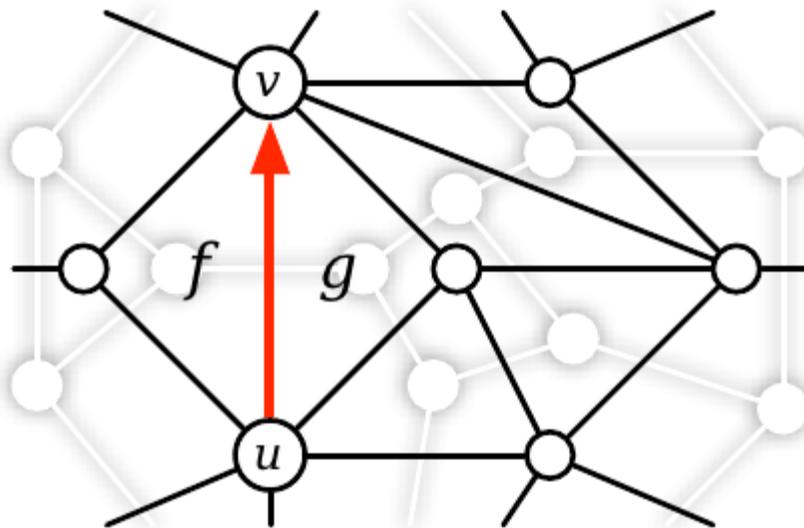
Idee:

- Betrachte kürzeste-Wege-Baum bzgl. $c(\lambda, \cdot)$ bei bel. Startknoten s .
- Erhöhe λ 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

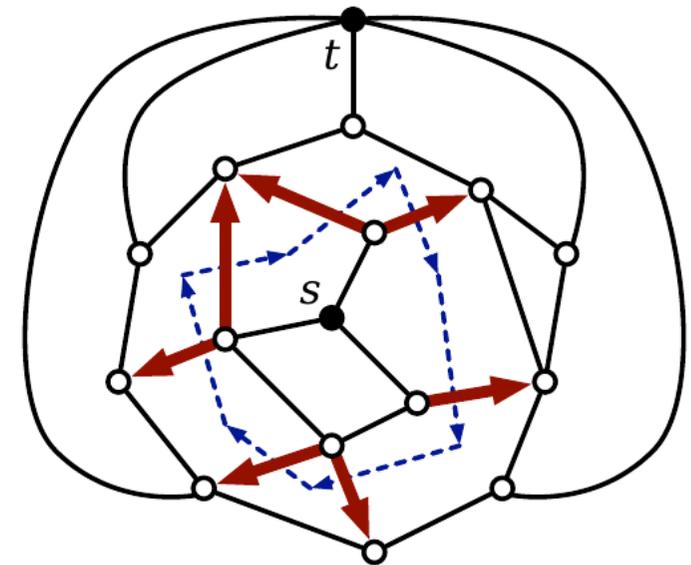
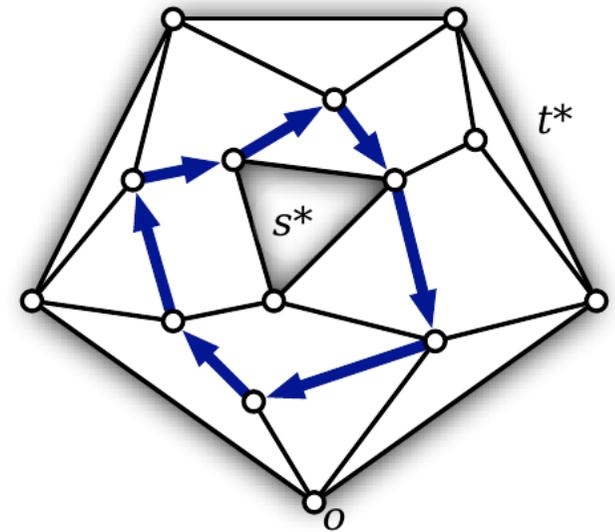
Dualgraph: Wie gehabt, $(u \rightarrow v)^*$ kreuzt $u \rightarrow v$ von links nach rechts!



[Erickson'10]

Gerichteter (s, t) -Schnitt:

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



Gerichteter (s, t) -Schnitt:

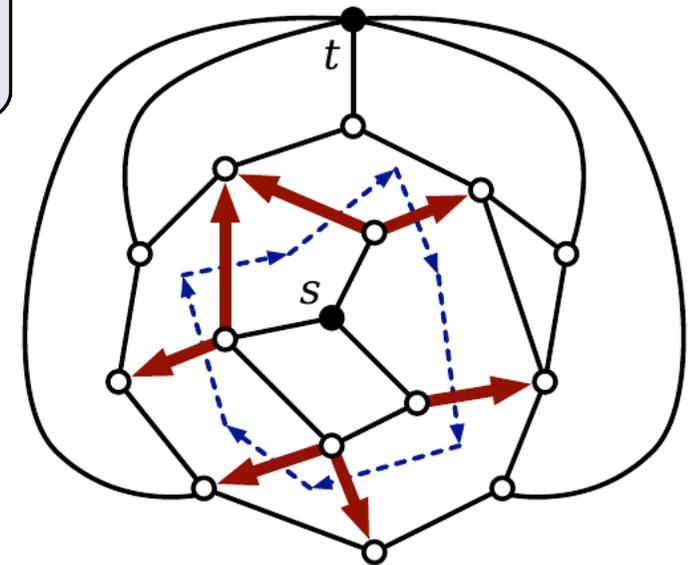
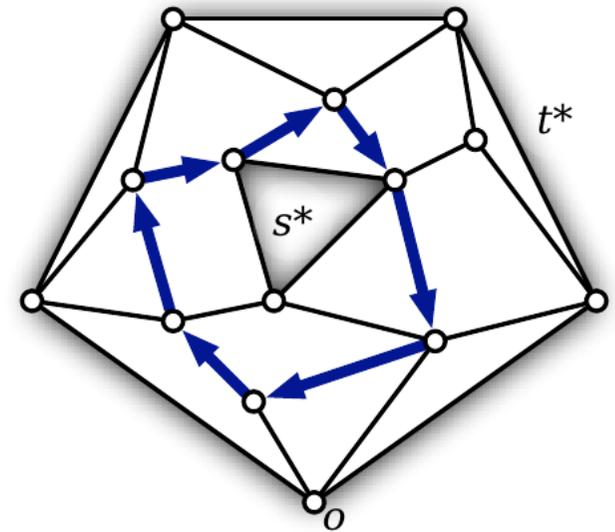
Kantenmenge C , sodass jeder gerichtete st -Pfad Kante aus C enthält.

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

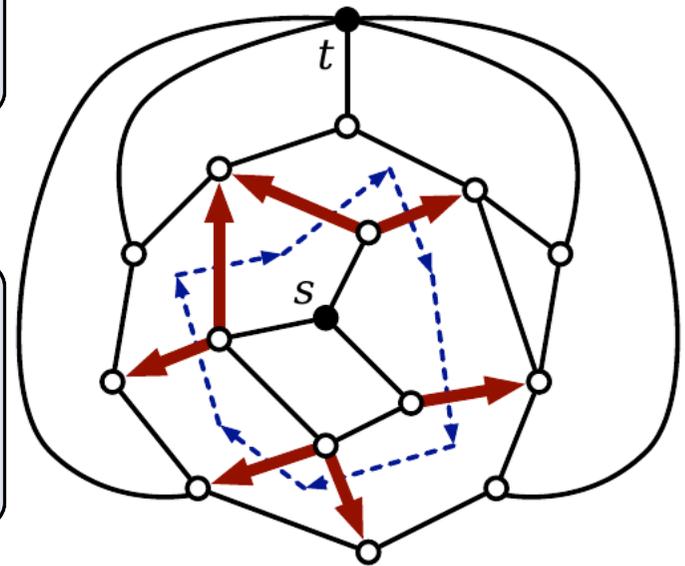
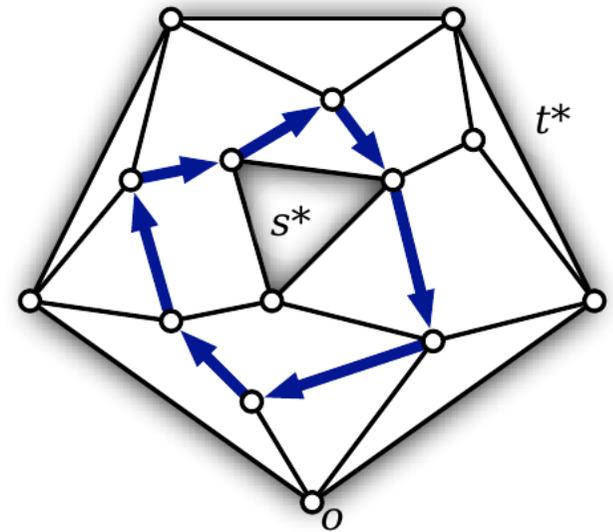
\equiv

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

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

$:\Leftrightarrow$

C^* einfacher gerichteter Kreis



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: $\pi(C) \in \{-1, 0, 1\}$

$\pi(C) = 1 \iff C$ ist (s, t) -Schnitt.

Betrachte Fluß von λ 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_λ^* : Dual-Graph mit e^* hat Kosten $c(\lambda, e^*) := c(\lambda, e)$.

Betrachte Fluß von λ 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_λ^* : Dual-Graph mit e^* hat Kosten $c(\lambda, e^*) := c(\lambda, e)$.

Lemma

G besitzt gültigen $s - t$ -Fluß mit Wert λ

\Leftrightarrow

G_λ^* enthält keinen negativen Kreis

Betrachte Fluß von λ 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_λ^* : Dual-Graph mit e^* hat Kosten $c(\lambda, e^*) := c(\lambda, e)$.

Lemma

G besitzt gültigen $s - t$ -Fluß mit Wert λ

\Leftrightarrow

G_λ^* enthält keinen negativen Kreis

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

Graph G **st -planar** wenn G sich so einbetten lässt, dass s und t an gemeinsamer Facette.

Satz

Ein maximaler st -Fluss in einem st -planaren Graphen kann in $O(n \log n)$ Zeit berechnet werden.

Graph G **st -planar** wenn G sich so einbetten lässt, dass s und t an gemeinsamer Facette.

Satz

Ein maximaler st -Fluss in einem st -planaren Graphen kann in $O(n \log n)$ Zeit berechnet werden.

In planaren Graphen können kürzeste Wege in $O(n)$ Zeit berechnet werden. [Henzinger et al. '97]

⇒ Laufzeit lässt sich auf $O(n)$ verbessern.

Lemma

G besitzt gültigen $s - t$ -Fluß mit Wert λ



G_λ^* enthält keinen negativen Kreis

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

Lemma

G besitzt gültigen $s - t$ -Fluß mit Wert λ

\Leftrightarrow

G_λ^* enthält keinen negativen Kreis

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

Verwalte Kürzeste-Wege-Baum T_λ in G_λ^* mit Wurzel o .

PlanarMaxFlow(G, c, s, t)

- Berechnet T_0
- Verwalte T_λ , während λ kontinuierlich von 0 bis λ_{\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 $\lambda \quad :\Leftrightarrow \quad \text{slack}(\lambda, e^*) = 0$.

Außer bei kritischen λ -Werten: e^* straff $\Leftrightarrow e^*$ in T_λ

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 $\lambda \quad :\Leftrightarrow \quad \text{slack}(\lambda, e^*) = 0$.

Außer bei kritischen λ -Werten: e^* straff $\Leftrightarrow e^*$ in T_λ

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 λ -Werten: e^* straff $\iff e^*$ in T_λ

Bei kritischen Werten:

Duale Nichtbaumkante $p \rightarrow q$ wird straff.

Pivotschritt

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

Lemma

λ_{\max} ist erster kritischer Wert von λ , dessen Pivot einen gerichteten Kreis in T_λ 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 λ -Werten: e^* straff $\iff e^*$ in T_λ

Bei kritischen Werten:

Duale Nichtbaumkante $p \rightarrow q$ wird straff.

Pivotschritt

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

Lemma

λ_{\max} ist erster kritischer Wert von λ , dessen Pivot einen gerichteten Kreis in T_λ erzeugt.

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

L_λ : lockere Kanten

Außer bei kritischen Werten: L_λ 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 λ -Werten: e^* straff $\iff e^*$ in T_λ

Bei kritischen Werten:

Duale Nichtbaumkante $p \rightarrow q$ wird straff.

Pivotschritt

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

Lemma

λ_{\max} ist erster kritischer Wert von λ , dessen Pivot einen gerichteten Kreis in T_λ erzeugt.

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

L_λ : lockere Kanten

Außer bei kritischen Werten: L_λ Spannbaum von G .

Lemma

λ_{\max} ist erster kritischer Wert von λ , dessen Pivot L_λ unzusammenhängend macht.

Duale Kante **aktiv** wenn slack mit λ abnimmt.

L_λ enthält eindeutigen st -Pfad LP_λ .

Lemma

Duale Kante e^* aktiv $\Leftrightarrow e$ Kante von LP_λ .

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_λ .

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.