

- MaxFlow in bipartiten Graphen
 - " in planaren Graphen
 - MinCut: wdh. Nagamochi-Ibaraki Implementation
-

PreFlow: Laufzeit:

$$\# \text{diff} \in O(n^2)$$

$$\# \text{sat. push} \in O(nm)$$

$$\# \text{non-sat. push} \in O(n^2m)$$

• Kritisch für Laufzeit!

Beweis lief über Potentialfunktion

$$\phi := \sum_{v \text{ aktiv}} h(v)$$

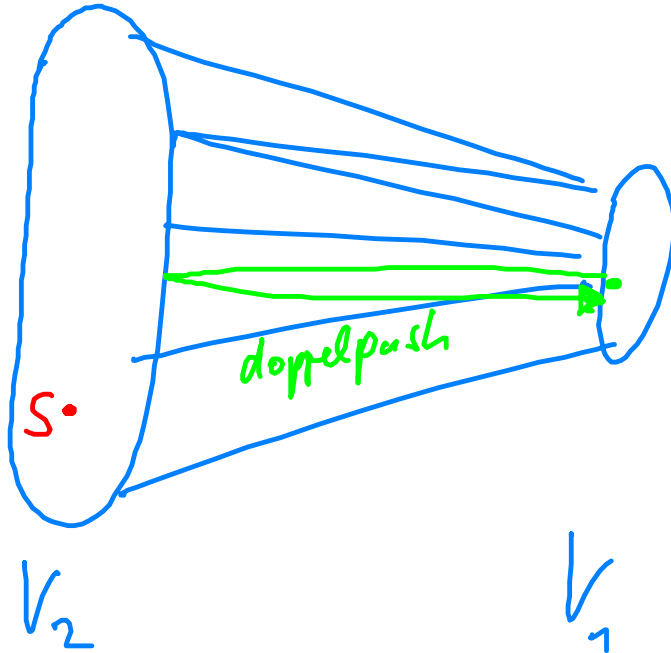
Sei $G = (V_1 \cup V_2, E)$ bipartit.

Sei $|V_1| = n_1$, $|V_2| = n_2$, o.B.d.A. $n_1 \leq n_2$

Sei o.B.d.A. $s \in V_2$ (falls $s \in V_1$, setze
 $V_2 := V_2 \cup \{s'\}$)

$E := E \cup \{(s', s)\}$
 $u(s', s)$ ausreichend groß)

Idee: Erlaube nur Knoten in V_1 , aktiv zu sein.



Init macht nur Nachbarn von s aktiv
 (alle in V_1). Dann "Doppelpush s ".

ALGO: • Init: saturiere alle $e \in \delta^+(s)$
 $h(s) = 2n_1 + 1, h(v) = 0$ für $v \neq s$

• while \exists aktives $v \in V \setminus \{s\}$
 pushLift(v);
 }

pushLift(v) {

if (v hat zul. Kante (v, w)) {

if (w hat zul. Kante (w, x))

augmentiere entlang (v, w), (w, x)

um mit $\{\varepsilon(v), u(v, w), u(w, x)\}$

else

lift(w);

}

else {

lift(v);

}

}

Lemma: $\# \text{ lift} \in O(n_1 \cdot n_2)$

Beweis: • Beim letzten pushLift(v) \exists v-s-Weg

P. $|P| \leq 2n_1$

• $h(s) = 2n_1 + 1$

• P hat nur zul. Kante

$\Rightarrow h(v) \leq 4n_1 + 1 \quad \forall v \in V$

$$\Rightarrow \# \text{ lift} \in O((u_1 + u_2)u_1) = O(u_1 u_2) \quad \square$$

Lemma: $\# \text{ sat. push} \in O(u_1 m)$

Beweis: • $h(v)$ erhöht sich zwischen zwei sat. push um ≤ 2

$$\bullet h(v) \leq 4u_1 + 1$$

• $\# \text{ sat. push je Kante} \in O(u_1) \quad \square$

Lemma: $\# \text{ non-sat. push} \in O(u_1^2 m)$

Beweis: $\phi := \sum_{v \text{ aktiv}} h(v)$

$$\Rightarrow \phi \leq 4u_1^2 (+ u_1)$$

Effekt von push(lift(v, w, x)) auf ϕ

1) lift(v) $\Rightarrow \phi \nearrow 1$ je Aufruf \Rightarrow gesamt $\nearrow O(u_1^2)$

2) lift(w) $\Rightarrow \phi \rightarrow$

3) sat. push $\Rightarrow \phi \nearrow 4u_1$ je Aufruf \Rightarrow gesamt $\nearrow O(u_1^2 m)$
(x wird aktiv!)

4) non-sat. push $\Rightarrow \phi \searrow 2$ je Aufruf

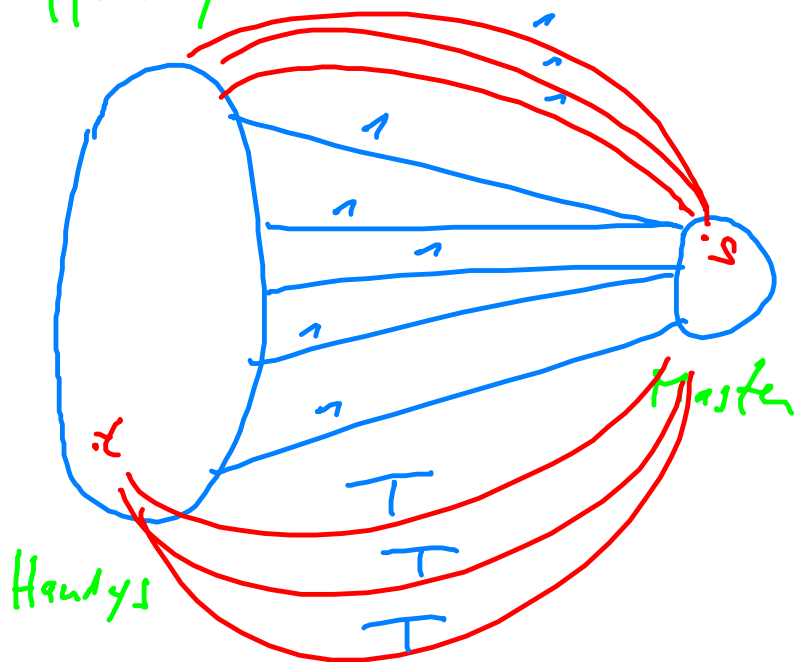
$$(h(v) - h(x)) \quad \square$$

SATZ: In bipartiten Graphen kann ein maximaler Fluss in $O(n^2 m)$ bestimmt werden. n bezeichnet die Kardinalität der kleineren Bipartition.

Also: Falls sich die Partitionen stark in ihrer Kardinalität unterscheiden, ist das besser als normale Preflow-ALP.

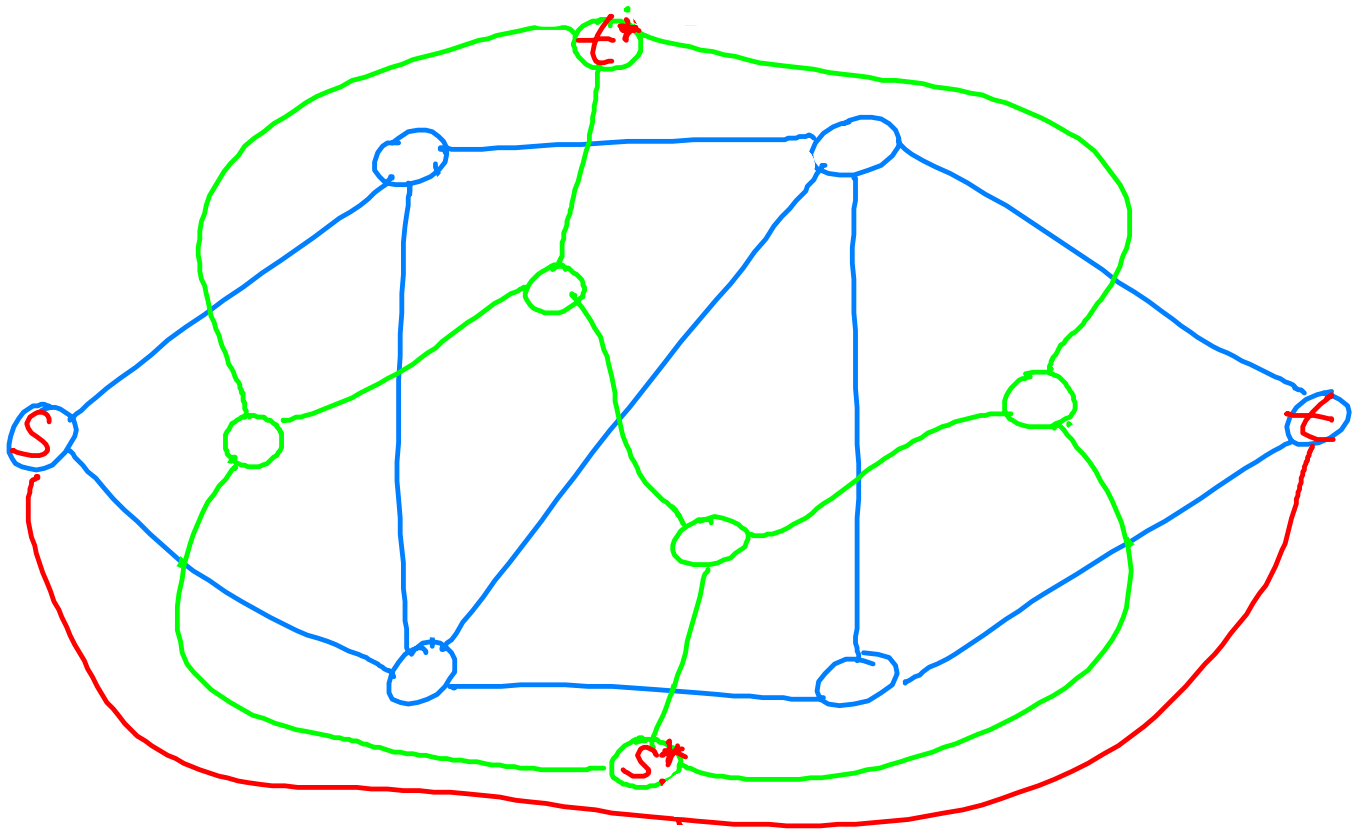
Bem.: Highest-Path findet zu $O(n^2 \sqrt{m})$.

Bsp.: Handys vs. Sendemasten



MaxFlow in unger. s - t -planaren Graphen

Def.: G s - t -planar falls G planar und s und t liegen auf der Grenze der äußeren Fläche.



1) Min Cut mit kürzeste Wege

1) Für u, v (s, t) in G ein

2) Dualgraph G^* ; bezeichne Knoten s^* t^*
(Knoten in äußerer Fläche)

3) Lösch $(s^*, t^*$ aus G^*)

Beobachtung: Kanten von G, G^* korrespondieren 1:1
 \Rightarrow identifiziere Kapazitäten

• $s^* - t^*$ -Weg in G^* korrespondieren 1:1
mit $s - t$ -Schritte in G

\Rightarrow minimaler $s - t$ -Schritt in G durch
kürzeste $s^* - t^*$ -Weg in G^*

Laufzeit: $O(m \log n) \stackrel{\text{planar}}{=} O(n \log n)$

(vorher $O(n \cdot m \cdot \log n)$ mit Nag.-Ibaraki)

2) MaxFlow aus kürz. Weglängen

Seien $d(v^*)$ die kürz. Weglänge von s^* nach v^*
in G^*

Definiere $x_{ij} := d(j^*) - d(i^*)$

Beh.: $f(i, j) := x_{ij}$ ist maximaler $s - t$ -Fluss

Beweis: 1) nicht-negativität:

Falls $x_{ij} < 0 \Rightarrow x_{ji} := -x_{ij} \checkmark$

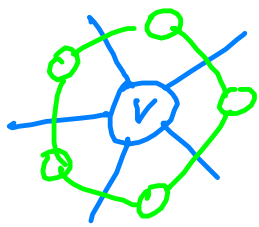
2) Kapazitäten

Es gilt: $d(j^*) \leq d(i^*) + u_{ij}$ falls $(i,j) \in E$

$$\Rightarrow x_{ij} = d(j^*) - d(i^*) \leq u_{ij} \quad \checkmark$$

3) Flussserhaltung

Betrachte für $v \neq s, t$ den Schnitt $Q := \delta(v)$



$\Rightarrow Q$ korrespondieren zu Kreis Z in G^*

$$\sum_{(i,j) \in Q} x_{ij} = \sum_{(i^*,j^*) \in Z} d(j^*) - d(i^*) = 0 \quad \checkmark$$

4) Optimalität

Sei P^* ein kürz. $s^* - t^*$ -Weg.

\Rightarrow auf P^* gilt $d(j^*) = d(i^*) + u_{ij} \quad (i^*,j^*) \in P^*$

$$\Rightarrow x_{ij} = d(j^*) - d(i^*) = u_{ij}$$

\Rightarrow alle Kante auf dem zu P^* korr. Schnitt sind saturiert \checkmark

SATZ: In unger. s-t-planare Graphen kann ein maximaler s-t-Fluss in $O(n \log n)$ bestimmt werden (vorher $O(n^2 \sqrt{m})$).

Fazit: Für jedes Problem lohnt es sich nochmal genau auf spezielle Graphenklasse anzusehen!

Nagamochi & Ibaraki

("Node Identification Algorithm")

- zur Bestimmung von Mächtig in unger. Graphen
- Ein maximum adjacency ordering (auch "legal ordering" ist eine Ordnung v_1, v_2, \dots, v_n der Knoten eines unger. Graphen mit

$$u(\delta(\{v_1, \dots, v_{i-1}\}) \cap \delta(v_i)) \geq u(\delta(\{v_1, \dots, v_{i-1}\}) \cap \delta(v_j))$$

$$\forall 2 \leq i < j \leq n$$

• Nächste Kante hat "größte Adjazenz" zu bisheriger Kante unter allen noch übrigen Kanten"

SATZ: Sei v_1, \dots, v_n ein legal ordering.
 $\Rightarrow \delta(v_n)$ ein minimales $v_{n-1} - v_n$ -Schritt.

ALGO: For $i=1 \dots n-1$ {
• Finde legal ordering
• Merke $d_i := u(\delta(v_n))$
• Kontrahiere v_{n-1}, v_n
}
Gib minimales d_i und zugehörigen Schritt zurück.

Laufzeit: $O(n \cdot m \log n)$

Merke: Es können beim Kontrahieren parallele Kanten entstehen.

Implementation:

- Merke für "Superknoten", welche "normal" Knoten sie enthalten

- Merke für jedes d_i den (Supr)Knoten, der ihn definiert.
- Für Ausgang des Schritts: Die Knoten im SuprKnoten, der das minimale d_i liefert, sind die den min. Schritt definierende Knotenmax.

