Seminar on Graphs, Algorithms and Optimization: **A Fast Parametric Maximum Flow Algorithm and Applications**

(Gallo, Grigoriadis, Tarjan, 1989)

Sandro M. Roch

02.07.2020

Parametric Maximum Flow Problem

Given: Directed Graph (V, E)
Source $s \in V$, Sink $t \in V$.
Capacities $c_\lambda(v, w) \geq 0$ for all $(v, w) \in E$,
$c_\lambda(v, w) = 0$ for all $(v, w) \notin E$, $\lambda \in \mathbb{R}$

Parametric Maximum Flow Problem

Given: Directed Graph (V, E)
Source $s \in V$, Sink $t \in V$.
Capacities $c_\lambda(v, w) \geq 0$ for all $(v, w) \in E$,
$c_\lambda(v, w) = 0$ for all $(v, w) \notin E$, $\lambda \in \mathbb{R}$

Problem for $\lambda \in \mathbb{R}$:

$$\kappa(\lambda) = \max \sum_{v \in V} f(v, t)$$

$$\begin{aligned}
\text{s.t.} \quad & f(v, w) = -f(w, v) && \text{for all } (v, w) \in V \times V && \text{(antisymmetry)} \\
& f(v, w) \leq c_\lambda(v, w) && \text{for all } (v, w) \in V \times V && \text{(capacity)} \\
& \sum_{w \in V} f(w, v) = 0 && \text{for all } v \in V \setminus \{s, t\} && \text{(conservation)}
\end{aligned}$$

Parametric Maximum Flow Problem

Given: Directed Graph (V, E)
Source $s \in V$, Sink $t \in V$.
Capacities $c_\lambda(v, w) \geq 0$ for all $(v, w) \in E$,
$c_\lambda(v, w) = 0$ for all $(v, w) \notin E$, $\lambda \in \mathbb{R}$

Problem for $\lambda \in \mathbb{R}$:

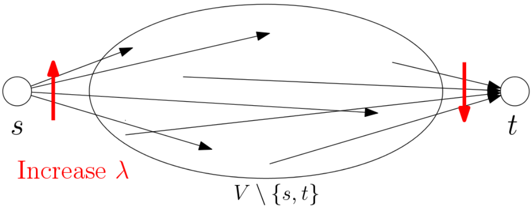
$$\kappa(\lambda) = \max \sum_{v \in V} f(v, t)$$

$$\begin{aligned}
\text{s.t.} \quad & f(v, w) = -f(w, v) && \text{for all } (v, w) \in V \times V && \text{(antisymmetry)} \\
& f(v, w) \leq c_\lambda(v, w) && \text{for all } (v, w) \in V \times V && \text{(capacity)} \\
& \sum_{w \in V} f(w, v) = 0 && \text{for all } v \in V \setminus \{s, t\} && \text{(conservation)}
\end{aligned}$$

Problem class too general to expect interesting parametric algorithm
$\Rightarrow$ Will restrict to subclass

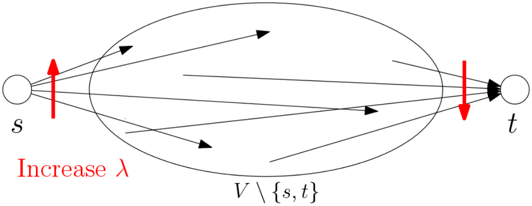
Restrictions on parametric capacities:

- $\lambda \mapsto c_\lambda(s, v)$ non-decreasing for all $v \in V$
- $\lambda \mapsto c_\lambda(v, t)$ non-increasing for all $v \in V$
- $c_\lambda(v, w)$ constant for all $v, w \in V \setminus \{s, t\}$



Restrictions on parametric capacities:

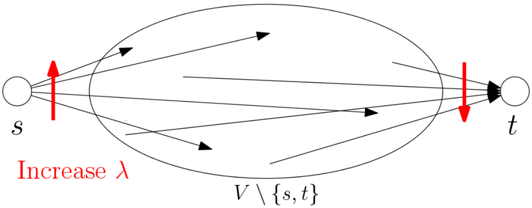
- $\lambda \mapsto c_\lambda(s, v)$ non-decreasing for all $v \in V$
- $\lambda \mapsto c_\lambda(v, t)$ non-increasing for all $v \in V$
- $c_\lambda(v, w)$ constant for all $v, w \in V \setminus \{s, t\}$



- Economic interpretation: Balance capacity budget between source incident arcs and sink incident arcs.

Restrictions on parametric capacities:

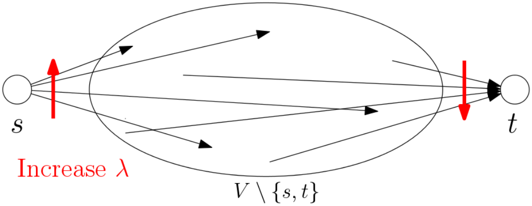
- $\lambda \mapsto c_\lambda(s, v)$ non-decreasing for all $v \in V$
- $\lambda \mapsto c_\lambda(v, t)$ non-increasing for all $v \in V$
- $c_\lambda(v, w)$ constant for all $v, w \in V \setminus \{s, t\}$



- Economic interpretation: Balance capacity budget between source incident arcs and sink incident arcs.
- Many algorithms for MaxFlow with fixed λ: Simplex-Algorithm, Ford-Fulkerson, Goldberg-Tarjan, Orlin, ...

Restrictions on parametric capacities:

- $\lambda \mapsto c_\lambda(s, v)$ non-decreasing for all $v \in V$
- $\lambda \mapsto c_\lambda(v, t)$ non-increasing for all $v \in V$
- $c_\lambda(v, w)$ constant for all $v, w \in V \setminus \{s, t\}$



- Economic interpretation: Balance capacity budget between source incident arcs and sink incident arcs.
- Many algorithms for MaxFlow with fixed λ: Simplex-Algorithm, Ford-Fulkerson, Goldberg-Tarjan, Orlin, ...
- **Outline:** Modify Goldberg-Tarjan to obtain parametric algorithm that calculates $\kappa(\lambda)$ for $\lambda_1 \leq ... \leq \lambda_l$ at once.

Definition

- A *preflow* $f : V \times V \to \mathbb{R}$ satisfies

$$
\begin{aligned}
f(v,w) &= -f(w,v) && \text{for all } (v,w) \in V \times V && \text{(antisymmetry)}\\
f(v,w) &\leq c_\lambda(v,w) && \text{for all } (v,w) \in V \times V && \text{(capacity)}\\
\sum_{w \in V} f(w,v) &\geq 0 && \text{for all } v \in V \setminus \{s\} && \text{(leaky conservation)}
\end{aligned}
$$

Revision of the Goldberg-Tarjan / Push-relabel algorithm

Definition

- A *preflow* $f : V \times V \to \mathbb{R}$ satisfies

$$f(v,w) = -f(w,v) \quad \text{for all } (v,w) \in V \times V \qquad \text{(antisymmetry)}$$
$$f(v,w) \leq c_\lambda(v,w) \quad \text{for all } (v,w) \in V \times V \qquad \text{(capacity)}$$
$$\sum_{w \in V} f(w,v) \geq 0 \quad \text{for all } v \in V \setminus \{s\} \qquad \text{(leaky conservation)}$$

- Under f a node $v \in V \setminus \{s,t\}$ is *active*, if $\sum_{w \in V} f(w,v) > 0$.

Revision of the Goldberg-Tarjan / Push-relabel algorithm

Definition

- A *preflow* $f : V \times V \to \mathbb{R}$ satisfies

$$\begin{aligned} f(v,w) &= -f(w,v) && \text{for all } (v,w) \in V \times V && \text{(antisymmetry)} \\ f(v,w) &\leq c_\lambda(v,w) && \text{for all } (v,w) \in V \times V && \text{(capacity)} \\ \sum_{w \in V} f(w,v) &\geq 0 && \text{for all } v \in V \setminus \{s\} && \text{(leaky conservation)} \end{aligned}$$

- Under f a node $v \in V \setminus \{s,t\}$ is *active*, if $\sum_{w \in V} f(w,v) > 0$.
- A *flow* is a preflow without active nodes.

Revision of the Goldberg-Tarjan / Push-relabel algorithm

Definition

- A *preflow* $f : V \times V \to \mathbb{R}$ satisfies

$$f(v,w) = -f(w,v) \quad \text{for all } (v,w) \in V \times V \qquad \text{(antisymmetry)}$$

$$f(v,w) \leq c_\lambda(v,w) \quad \text{for all } (v,w) \in V \times V \qquad \text{(capacity)}$$

$$\sum_{w \in V} f(w,v) \geq 0 \quad \text{for all } v \in V \setminus \{s\} \qquad \text{(leaky conservation)}$$

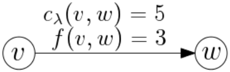
- Under f a node $v \in V \setminus \{s,t\}$ is *active*, if $\sum_{w \in V} f(w,v) > 0$.
- A *flow* is a preflow without active nodes.

Remark: Flows are the feasible solutions of the LP (*conservation* instead of *leaky conservation*)

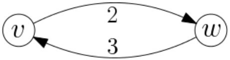
Definition

- Given preflow f. If $f(v,w) < c_\lambda(v,w)$, then (v,w) is called *residual arc* with *residual capacity* $c_\lambda(v,w) - f(v,w)$.
- $d_f(v,w)$: Number of arcs of shortest residual path from v to w or ∞.

Example:



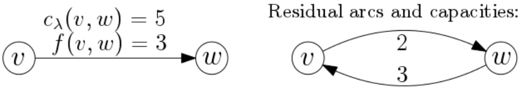
Residual arcs and capacities:



Definition

- Given preflow f. If $f(v,w) < c_\lambda(v,w)$, then (v,w) is called *residual arc* with *residual capacity* $c_\lambda(v,w) - f(v,w)$.
- $d_f(v,w)$: Number of arcs of shortest residual path from v to w or ∞.

Example:



Definition

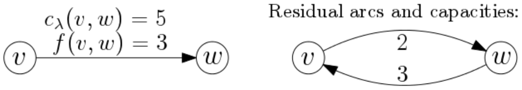
For preflow f, assignment $d : V \to \mathbb{N}$ is *valid labeling* if $d(s) = n$, $d(t) = 0$ and $d(v) \leq d(w) + 1$ for all residual arcs (v,w).

- No sharp descents of valid labels d on residual arcs

Definition

- Given preflow f. If $f(v,w) < c_\lambda(v,w)$, then (v,w) is called *residual arc* with *residual capacity* $c_\lambda(v,w) - f(v,w)$.
- $d_f(v,w)$: Number of arcs of shortest residual path from v to w or ∞.

Example:



Definition

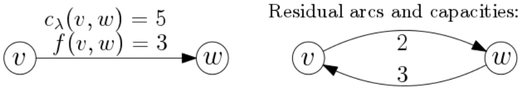
For preflow f, assignment $d : V \to \mathbb{N}$ is *valid labeling* if $d(s) = n$, $d(t) = 0$ and $d(v) \leq d(w) + 1$ for all residual arcs (v,w).

- No sharp descents of valid labels d on residual arcs
- For all nodes v: $d(v) \leq \min\{d_f(v,t), d_f(v,s) + n\}$

Definition

- Given preflow f. If $f(v,w) < c_\lambda(v,w)$, then (v,w) is called *residual arc* with *residual capacity* $c_\lambda(v,w) - f(v,w)$.
- $d_f(v,w)$: Number of arcs of shortest residual path from v to w or ∞.

Example:



Definition

For preflow f, assignment $d : V \to \mathbb{N}$ is *valid labeling* if $d(s) = n$, $d(t) = 0$ and $d(v) \leq d(w) + 1$ for all residual arcs (v,w).

- No sharp descents of valid labels d on residual arcs
- For all nodes v: $d(v) \leq \min\{d_f(v,t), d_f(v,s) + n\}$
- For all active nodes v: $\min\{d_f(v,t), d_f(v,s) + n\} \leq 2n - 1$

Claim: Flow f has valid labeling $d \Rightarrow f$ is maximal flow.

- If f not maximal: Exists residual path from s to t of length $\leq n-1$
- Then d is no valid labeling, since $d(s) = n$ and $d(t) = 0$ cannot hold.

Claim: Flow f has valid labeling $d \Rightarrow f$ is maximal flow.

- If f not maximal: Exists residual path from s to t of length $\leq n-1$
- Then d is no valid labeling, since $d(s) = n$ and $d(t) = 0$ cannot hold.

Goldberg-Tarjan algorithm

- **Initialize** preflow f and valid labeling d.
- As long as exist active nodes:
  Eliminate active node by push-relabel-operation
  (might create new active nodes).
- **Finally:** f is flow, d is still valid labeling.

Claim: Flow f has valid labeling $d \Rightarrow f$ is maximal flow.

- If f not maximal: Exists residual path from s to t of length $\leq n - 1$
- Then d is no valid labeling, since $d(s) = n$ and $d(t) = 0$ cannot hold.

Goldberg-Tarjan algorithm

- **Initialize** preflow f and valid labeling d.
- As long as exist active nodes:
  Eliminate active node by push-relabel-operation
  (might create new active nodes).
- **Finally:** f is flow, d is still valid labeling.

Remarks on Goldberg-Tarjan:

- Throughout execution: Labeling d remains valid and never decreases.

Claim: Flow f has valid labeling $d \Rightarrow f$ is maximal flow.

- If f not maximal: Exists residual path from s to t of length $\leq n - 1$
- Then d is no valid labeling, since $d(s) = n$ and $d(t) = 0$ cannot hold.

Goldberg-Tarjan algorithm

- **Initialize** preflow f and valid labeling d.
- As long as exist active nodes:
  Eliminate active node by push-relabel-operation
  (might create new active nodes).
- **Finally:** f is flow, d is still valid labeling.

Remarks on Goldberg-Tarjan:

- Throughout execution: Labeling d remains valid and never decreases.
- In some push-relabel-operations, d must increase.

Claim: Flow f has valid labeling $d \Rightarrow f$ is maximal flow.

- If f not maximal: Exists residual path from s to t of length $\leq n-1$
- Then d is no valid labeling, since $d(s) = n$ and $d(t) = 0$ cannot hold.

Goldberg-Tarjan algorithm

- **Initialize** preflow f and valid labeling d.
- As long as exist active nodes:
  Eliminate active node by push-relabel-operation
  (might create new active nodes).
- **Finally:** f is flow, d is still valid labeling.

Remarks on Goldberg-Tarjan:

- Throughout execution: Labeling d remains valid and never decreases.
- In some push-relabel-operations, d must increase.
- Termination is due to $d(v) \leq 2n-1$ for active nodes v.

Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.

Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.

Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow“ to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).

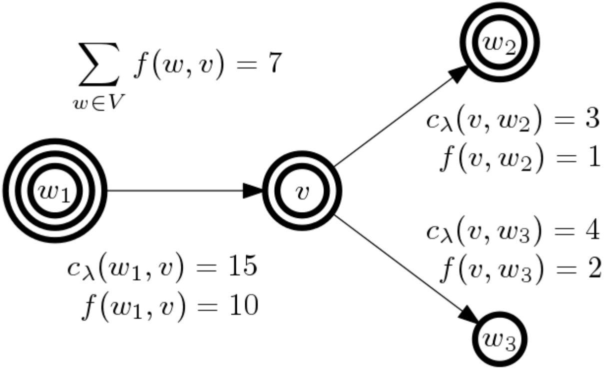
Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

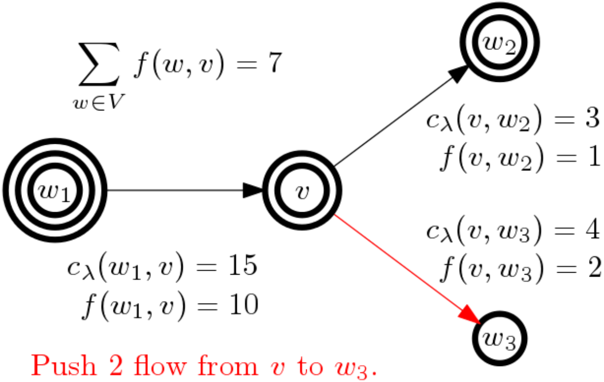
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow“ to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

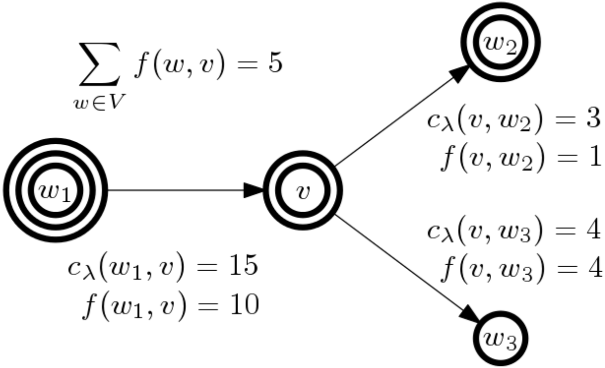
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

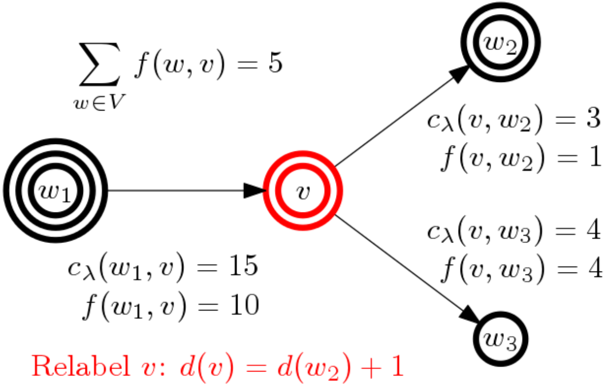
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

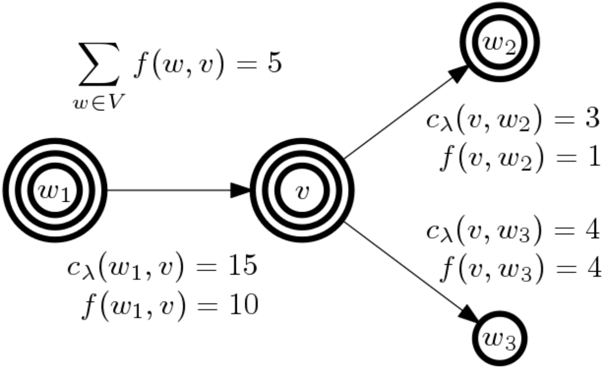
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

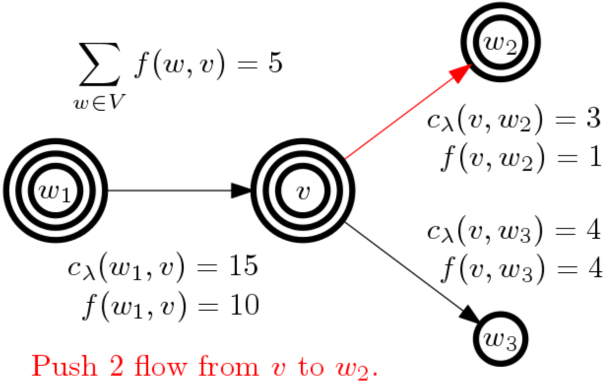
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow“ to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

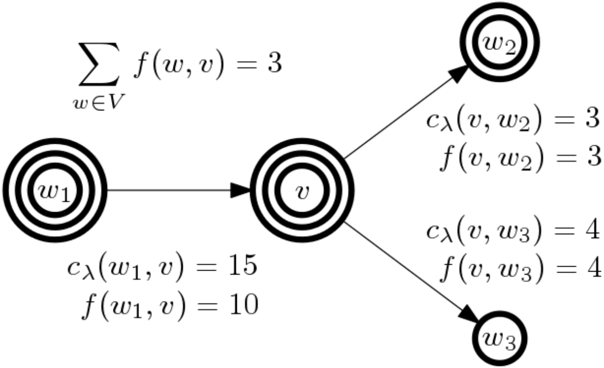
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

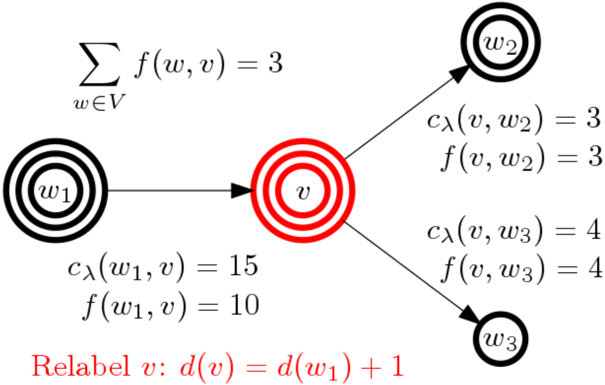
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

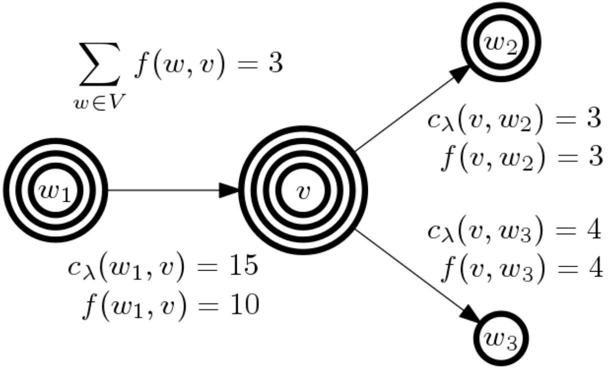
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

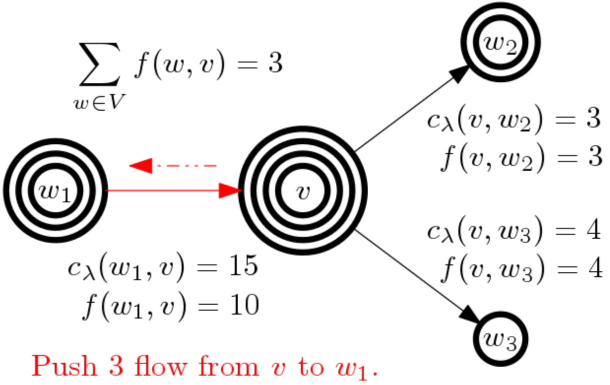
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

- **Goal:** Eliminate active node v by „pushing away flow" to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

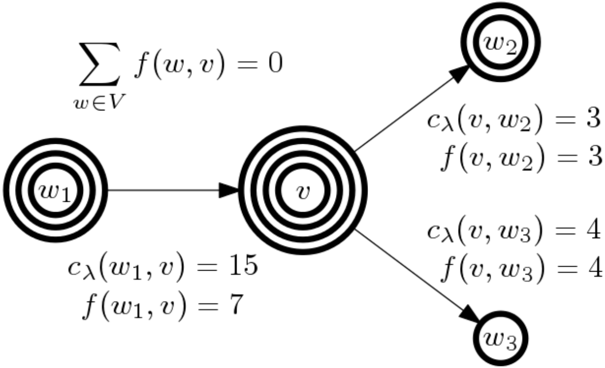
Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



Push-relabel-operation:

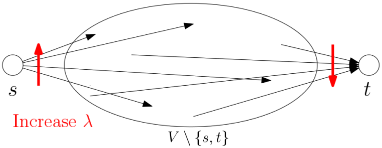
- **Goal:** Eliminate active node v by „pushing away flow“ to neighbours.
- But push flow only to neighbours $w \in N(v)$ with $d(w) < d(v)$.
- Therefore increase $d(v)$ as much as neccessary (*relabel*).
- **Finally:** v not active anymore and d still valid labeling

Example: Push-relabel on node v with $N(v) = \{w_1, w_2, w_3\}$:



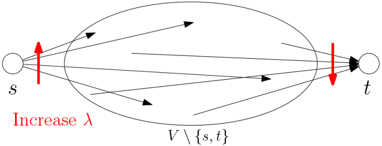
Goldberg Tarjan for two values of λ

- Want to calculate MaxFlow for $\lambda_1 \leq \lambda_2$



Goldberg Tarjan for two values of λ

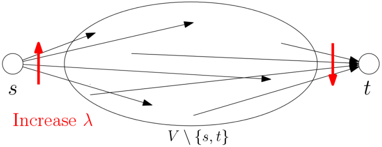
- Want to calculate MaxFlow for $\lambda_1 \leq \lambda_2$



- **Idea** of Parametric Goldberg Tarjan algorithm:
    - Run Goldberg Tarjan first for λ_1: MaxFlow f, valid labeling d

Goldberg Tarjan for two values of λ

- Want to calculate MaxFlow for $\lambda_1 \leq \lambda_2$

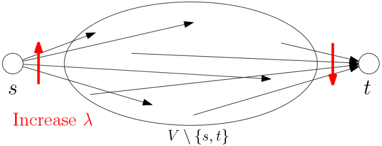


- **Idea** of Parametric Goldberg Tarjan algorithm:
  - Run Goldberg Tarjan first for λ_1: MaxFlow f, valid labeling d
  - Modify flow f for c_{λ_1} to obtain preflow f' for c_{λ_2}:

$$f'(v,w) := \begin{cases} \min\{c_{\lambda_2}(v,t), f(v,t)\} & \text{if } w = t \\ \max\{c_{\lambda_2}(s,w), f(s,w)\} & \text{if } v = s \text{ and } d(w) < n \\ f(v,w) & \text{otherwise} \end{cases}$$

Goldberg Tarjan for two values of λ

- Want to calculate MaxFlow for $\lambda_1 \leq \lambda_2$



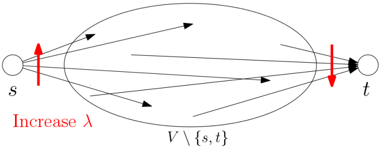
- **Idea** of Parametric Goldberg Tarjan algorithm:
  - Run Goldberg Tarjan first for λ_1: MaxFlow f, valid labeling d
  - Modify flow f for c_{λ_1} to obtain preflow f' for c_{λ_2}:

$$f'(v,w) := \begin{cases} \min\{c_{\lambda_2}(v,t), f(v,t)\} & \text{if } w = t \\ \max\{c_{\lambda_2}(s,w), f(s,w)\} & \text{if } v = s \text{ and } d(w) < n \\ f(v,w) & \text{otherwise} \end{cases}$$

  - Observe: f' is indeed preflow and d valid labeling for f'.

Goldberg Tarjan for two values of λ

- Want to calculate MaxFlow for $\lambda_1 \leq \lambda_2$



- **Idea** of Parametric Goldberg Tarjan algorithm:
  - Run Goldberg Tarjan first for λ_1: MaxFlow f, valid labeling d
  - Modify flow f for c_{λ_1} to obtain preflow f' for c_{λ_2}:

$$f'(v,w) := \begin{cases} \min\{c_{\lambda_2}(v,t), f(v,t)\} & \text{if } w = t \\ \max\{c_{\lambda_2}(s,w), f(s,w)\} & \text{if } v = s \text{ and } d(w) < n \\ f(v,w) & \text{otherwise} \end{cases}$$

  - Observe: f' is indeed preflow and d valid labeling for f'.
  - Run Goldberg Tarjan for λ_2, but start with f' and d.

Obtain MinCut from MaxFlow

- MinCut: $(X, X^{\complement})$ with $s \in X \subsetneq V$ and $\delta^+_{c_\lambda}(X)$ minimal

Obtain MinCut from MaxFlow

- MinCut: $(X, X^{\complement})$ with $s \in X \subsetneq V$ and $\delta^{+}_{c_\lambda}(X)$ minimal
- MaxFlow = MinCut (Ford & Fulkerson, 1962)

Obtain MinCut from MaxFlow

- MinCut: $(X, X^{\complement})$ with $s \in X \subsetneq V$ and $\delta^{+}_{c_\lambda}(X)$ minimal
- MaxFlow = MinCut (Ford & Fulkerson, 1962)

Obtain MinCut $(X, X^{\complement})$ **from MaxFlow** f**:**

- Goldberg-Tarjan returns MaxFlow f and valid labeling d

Obtain MinCut from MaxFlow

- MinCut: $(X, X^{\complement})$ with $s \in X \subsetneq V$ and $\delta^{+}_{c_\lambda}(X)$ minimal
- MaxFlow = MinCut (Ford & Fulkerson, 1962)

Obtain MinCut $(X, X^{\complement})$ from MaxFlow f:

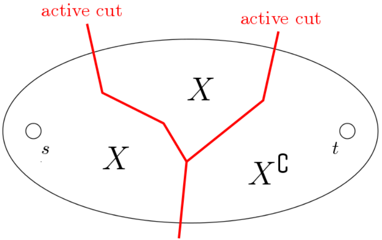
- Goldberg-Tarjan returns MaxFlow f and valid labeling d
- $X := \{v \in V \mid \nexists \text{ residual path from } v \text{ to } t\}$ is MinCut:
  - For all $w \in X^{\complement}$ exists residual path from w to t
  - Any arc $e \in X \times X^{\complement}$ cannot be residual arc.
  - $(X, X^{\complement})$ is „active cut“: $\forall e \in X \times X^{\complement} : f(e) = c_\lambda(e)$

Obtain MinCut from MaxFlow

- MinCut: $(X, X^\complement)$ with $s \in X \subsetneq V$ and $\delta^+_{c_\lambda}(X)$ minimal
- MaxFlow = MinCut (Ford & Fulkerson, 1962)

Obtain MinCut $(X, X^\complement)$ from MaxFlow f:

- Goldberg-Tarjan returns MaxFlow f and valid labeling d
- $X := \{v \in V \mid \nexists \text{ residual path from } v \text{ to } t\}$ is MinCut:
  - For all $w \in X^\complement$ exists residual path from w to t
  - Any arc $e \in X \times X^\complement$ cannot be residual arc.
  - $(X, X^\complement)$ is „active cut“: $\forall e \in X \times X^\complement : f(e) = c_\lambda(e)$



Parametric Goldberg Tarjan Algorithm

Input: Directed graph (V, E); $s, t \in V$; capacities c_λ; $\lambda_1 \leq ... \leq \lambda_l$
Output: MaxFlow f_i and MinCut $(X_i, X_i^\complement)$ for $i = 1, ..., l$

Initialize $f = 0$; $d(s) \leftarrow n$; $d(v) \leftarrow 0$ f.a. $v \neq s$
for $i = 1, ..., l$ **do**

Step 1: Update preflow

$$f(v,w) \leftarrow \begin{cases} \min\{c_{\lambda_i}(v,t), f(v,t)\} & \text{if } w = t \\ \max\{c_{\lambda_i}(s,w), f(s,w)\} & \text{if } v = s \text{ and } d(w) < n \\ f(v,w) & \text{otherwise} \end{cases}$$

Step 2: Run Goldberg Tarjan
$(f, d) \leftarrow \text{GoldbergTarjan}(f, d)$

Step 3: Find MinCut
$d(v) \leftarrow \min\{d_f(v,s) + n, d_f(v,t)\}$ for $v \in V$
$X \leftarrow \{v \in V \mid d(v) \geq n\}$
Output „$X_i = X, f_i = f$“

end

Analysis of Parametric Goldberg Tarjan

Theorem

1) *The Parametric Goldberg Tarjan Algorithm works correctly with running time* $\mathcal{O}(n^2(l+m))$.

Analysis of Parametric Goldberg Tarjan

Theorem

1) *The Parametric Goldberg Tarjan Algorithm works correctly with running time* $\mathcal{O}(n^2(l+m))$.
2) *The returned MinCuts* $(X_i, X_i^\complement)$ *are nested, i.e.* $X_1 \subseteq X_2 \subseteq ... \subseteq X_l$.

Analysis of Parametric Goldberg Tarjan

Theorem

1) *The Parametric Goldberg Tarjan Algorithm works correctly with running time* $\mathcal{O}(n^2(l+m))$.
2) *The returned MinCuts* $(X_i, X_i^\complement)$ *are nested, i.e.* $X_1 \subseteq X_2 \subseteq \ldots \subseteq X_l$.

Proof of 1):

- Correctness from discussion before
- Step 1 (Update preflow) requires $\mathcal{O}(m)$ per iteration
- Step 2 (Goldberg Tarjan) $\mathcal{O}(nm) + \#$*non-saturating pushs*
- Step 3 (Find MinCut) requires $\mathcal{O}(m)$ per iteration
- Σ: $\mathcal{O}((n+l)m) + \#$*non-saturating pushs*
- As in ADM I: $\#$*non-saturating pushs* $\in \mathcal{O}(n^2(l+m))$
- In total: Running time $\mathcal{O}(n^2(l+m))$

Analysis of Parametric Goldberg Tarjan

Theorem

1) *The Parametric Goldberg Tarjan Algorithm works correctly with running time* $\mathcal{O}(n^2(l+m))$.
2) *The returned MinCuts* $(X_i, X_i^\complement)$ *are nested, i.e.* $X_1 \subseteq X_2 \subseteq \ldots \subseteq X_l$.

Proof of 2):

- Throughout execution, d only increases.
- Cut is chosen as $X_i = \{v \in V \mid d(v) \geq n\}$ □

Corollary

If $l \in \mathcal{O}(n)$, the asymptotic running time of the Parametric Goldberg Tarjan algorithm is the same as for the normal Goldberg Tarjan algorithm.

Corollary

If $l \in \mathcal{O}(n)$, the asymptotic running time of the Parametric Goldberg Tarjan algorithm is the same as for the normal Goldberg Tarjan algorithm.

Proof:

- Parametric Goldberg Tarjan runs in $\mathcal{O}(n^2(l+m))$
- Since $l \in \mathcal{O}(n)$: Running time $\mathcal{O}(n^2 m)$ $\square$

Corollary

If $l \in \mathcal{O}(n)$, the asymptotic running time of the Parametric Goldberg Tarjan algorithm is the same as for the normal Goldberg Tarjan algorithm.

Proof:

- Parametric Goldberg Tarjan runs in $\mathcal{O}(n^2(l+m))$
- Since $l \in \mathcal{O}(n)$: Running time $\mathcal{O}(n^2 m)$ $\square$

Remark:

- Improve both to $\mathcal{O}((n+l)m \log \frac{n^2}{m})$ by advanced strategies for push-relabel operations and advanced data structures.
- Parametric Goldberg Tarjan algorithm works **on-line** for sequence $\lambda_1 \leq ... \leq \lambda_l$.

Maximization of $\kappa(\lambda)$

- **Goal:** Determine $\lambda \in \mathbb{R}$ so that $\lambda \mapsto \kappa(\lambda)$ maximal
- Assume $\lambda \mapsto c_\lambda(v, w)$ affine linear for all $(v, w) \in V \times V$

Maximization of $\kappa(\lambda)$

- **Goal:** Determine $\lambda \in \mathbb{R}$ so that $\lambda \mapsto \kappa(\lambda)$ maximal
- Assume $\lambda \mapsto c_\lambda(v, w)$ affine linear for all $(v, w) \in V \times V$

Corollary

For arbitrary many $\lambda_1 \leq ... \leq \lambda_l$ the Parametric Goldberg Tarjan algorithm outputs at most $n - 1$ distinct MinCuts $(X_i, X_i^\complement)$.

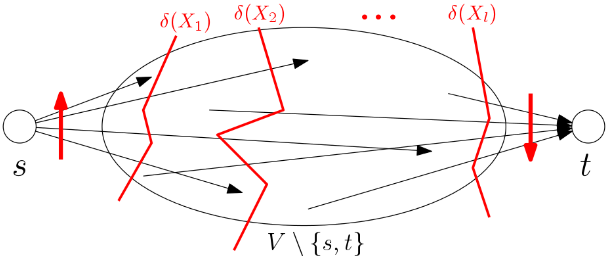
Maximization of $\kappa(\lambda)$

- **Goal:** Determine $\lambda \in \mathbb{R}$ so that $\lambda \mapsto \kappa(\lambda)$ maximal
- Assume $\lambda \mapsto c_\lambda(v, w)$ affine linear for all $(v, w) \in V \times V$

Corollary

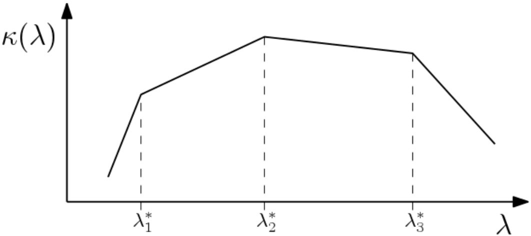
For arbitrary many $\lambda_1 \leq ... \leq \lambda_l$ the Parametric Goldberg Tarjan algorithm outputs at most $n - 1$ distinct MinCuts $(X_i, X_i^\complement)$.

Proof: Follows directly from nested property $X_1 \subseteq X_2 \subseteq ... \subseteq X_l$.



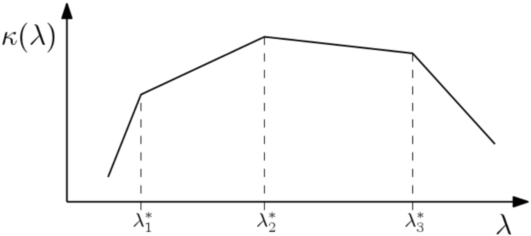
Recall from ADM II (Global Dependence on the Cost Vector):

- $\kappa(\lambda)$ is concave and piecewise linear.
- Breakpoints correspond to basis changes.



Recall from ADM II (Global Dependence on the Cost Vector):

- $\kappa(\lambda)$ is concave and piecewise linear.
- Breakpoints correspond to basis changes.



Consequence:

- Now basis change means change of MinCut $(X, X^{\complement})$.
- Therefore at most $n-1$ breakpoints.

Calculate the line segments of $\kappa(\lambda)$:

- Assume parametric capacities are given for $v \in V \setminus \{s, t\}$ by

$$c_\lambda(s, v) = a_0(v) + \lambda \cdot a_1(v)$$
$$c_\lambda(v, t) = b_0(v) - \lambda \cdot b_1(v)$$

with $a_0(v), b_0(v) \in \mathbb{R}$ and $a_1(v), b_1(b) \geq 0$.

Calculate the line segments of $\kappa(\lambda)$:

- Assume parametric capacities are given for $v \in V \setminus \{s, t\}$ by

$$c_\lambda(s, v) = a_0(v) + \lambda \cdot a_1(v)$$
$$c_\lambda(v, t) = b_0(v) - \lambda \cdot b_1(v)$$

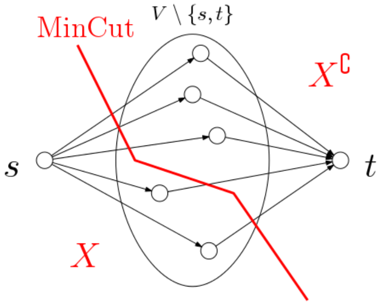
  with $a_0(v), b_0(v) \in \mathbb{R}$ and $a_1(v), b_1(b) \geq 0$.
- Assume $\lambda_0 \in \mathbb{R}$ is not a breakpoint and $(X, X^\complement)$ is MinCut for $\lambda = \lambda_0$.

Calculate the line segments of $\kappa(\lambda)$:

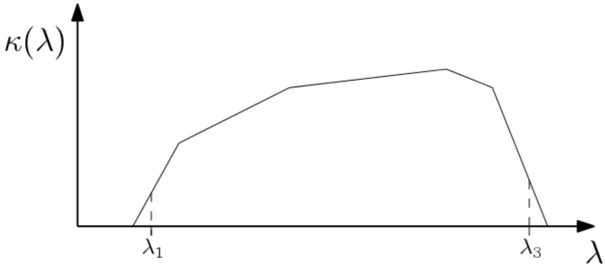
- Assume parametric capacities are given for $v \in V \setminus \{s, t\}$ by

$$c_\lambda(s, v) = a_0(v) + \lambda \cdot a_1(v)$$
$$c_\lambda(v, t) = b_0(v) - \lambda \cdot b_1(v)$$

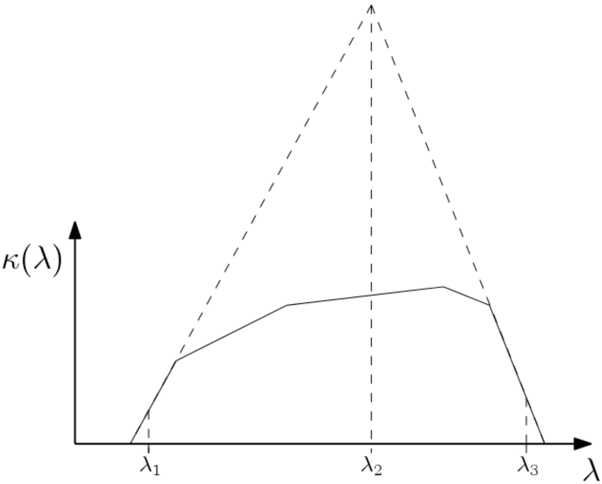
  with $a_0(v), b_0(v) \in \mathbb{R}$ and $a_1(v), b_1(b) \geq 0$.
- Assume $\lambda_0 \in \mathbb{R}$ is not a breakpoint and $(X, X^{\complement})$ is MinCut for $\lambda = \lambda_0$.
- Line segment of $\kappa(\lambda)$ around λ_0:
  $L(\lambda) = \kappa(\lambda_0) + (\lambda - \lambda_0) \cdot \left(\sum_{v \in X^{\complement} \setminus \{t\}} a_1(v) - \sum_{v \in X \setminus \{s\}} b_1(v)\right)$



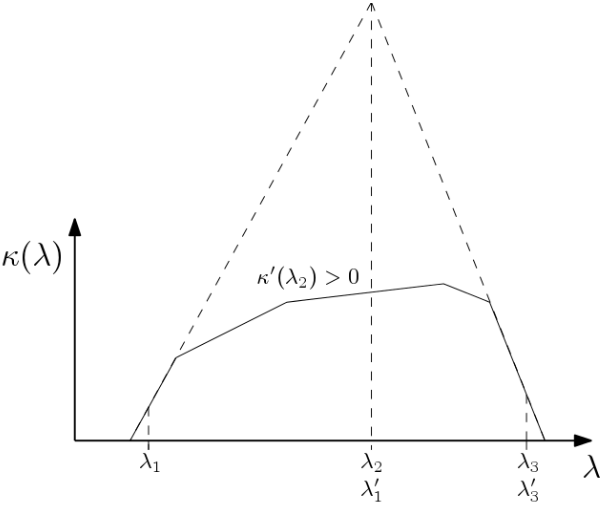
Example: Assume $\lambda_{\max} \in [\lambda_1, \lambda_3]$:



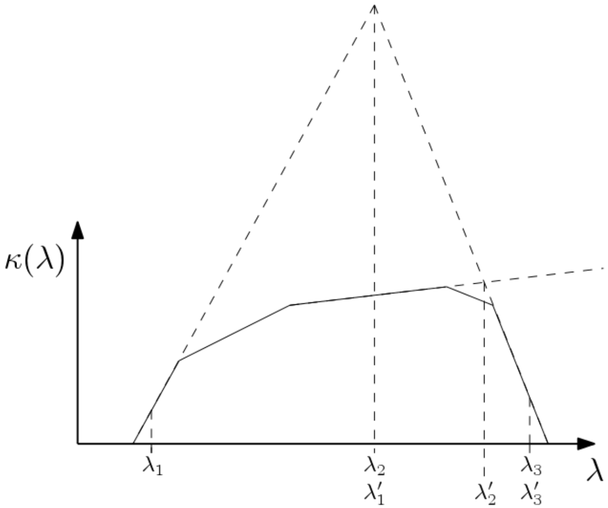
Example: Assume $\lambda_{\max} \in [\lambda_1, \lambda_3]$:



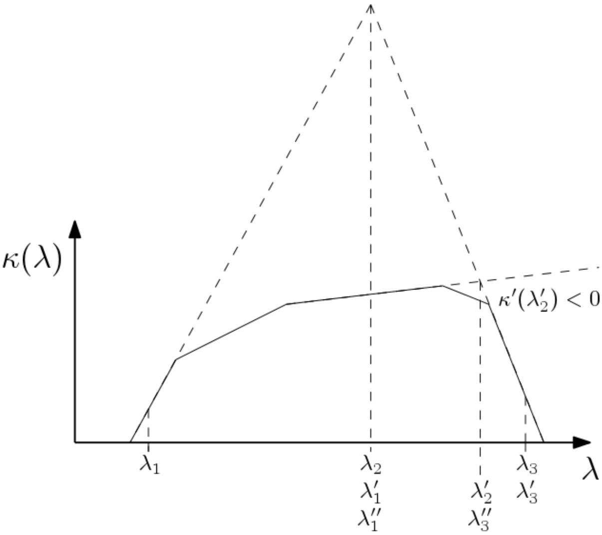
Example: Assume $\lambda_{\max} \in [\lambda_1, \lambda_3]$:



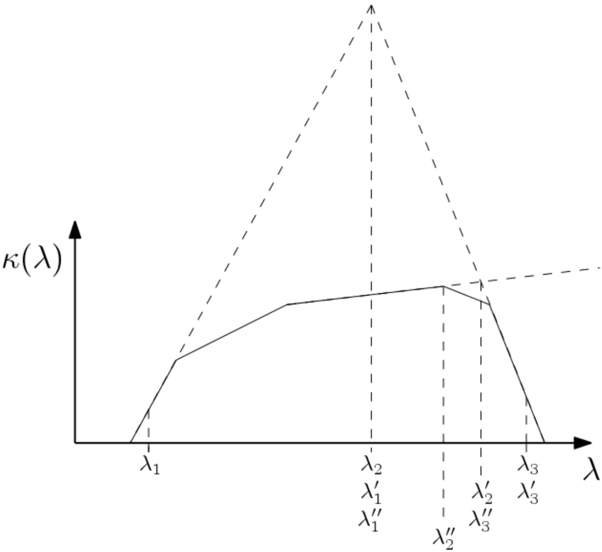
Example: Assume $\lambda_{\max} \in [\lambda_1, \lambda_3]$:



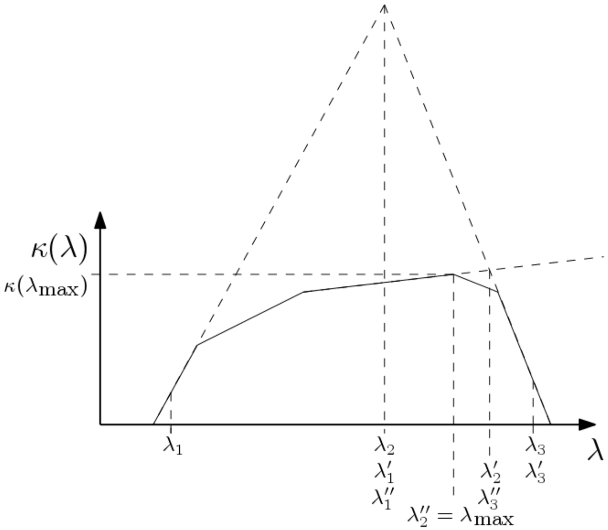
Example: Assume $\lambda_{\max} \in [\lambda_1, \lambda_3]$:

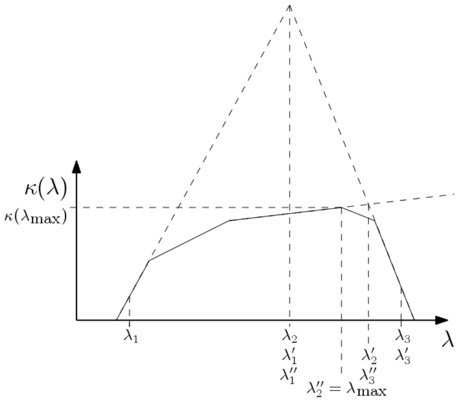


Example: Assume $\lambda_{\max} \in [\lambda_1, \lambda_3]$:



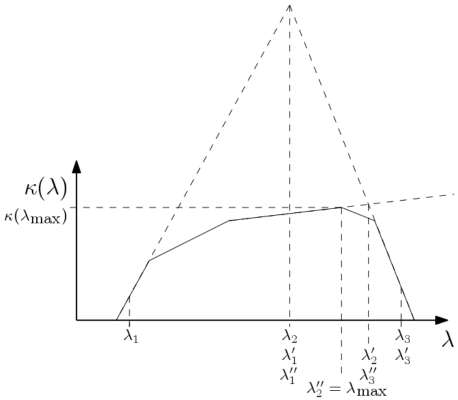
Example: Assume $\lambda_{\max} \in [\lambda_1, \lambda_3]$:





Basic idea:

- Shrink interval $[\lambda_1, \lambda_3]$ with $\lambda_{max} \in [\lambda_1, \lambda_3]$ as in example.

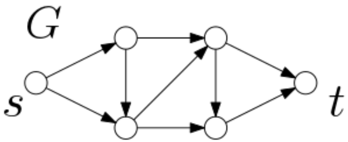


Basic idea:

- Shrink interval $[\lambda_1, \lambda_3]$ with $\lambda_{\max} \in [\lambda_1, \lambda_3]$ as in example.
- For calculating $\kappa(\lambda)$ for $\lambda_1 \leq \lambda_1' \leq \lambda_1'' \leq \ldots$ and $\lambda_3 \geq \lambda_3' \geq \lambda_3'' \geq \ldots$ use two concurrent runs of Parametric Goldberg Tarjan.

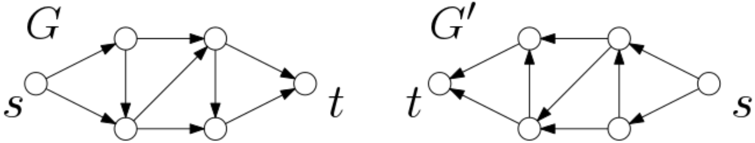
Adapt Parametric Goldberg Tarjan to decreasing values of λ

- **Idea:** From G obtain equivalent problem G' by reversing all arcs and interchanging source and sink.



Adapt Parametric Goldberg Tarjan to decreasing values of λ

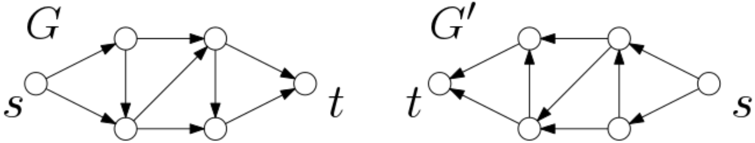
- **Idea:** From G obtain equivalent problem G' by reversing all arcs and interchanging source and sink.



- In G', $\lambda \mapsto c_\lambda(s, v)$ non-increasing.

Adapt Parametric Goldberg Tarjan to decreasing values of λ

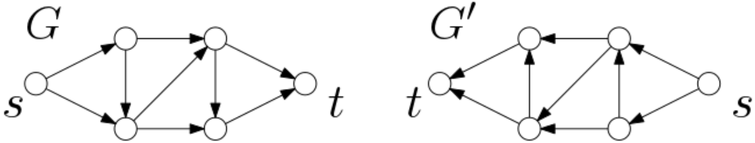
- **Idea:** From G obtain equivalent problem G' by reversing all arcs and interchanging source and sink.



- In G', $\lambda \mapsto c_\lambda(s, v)$ non-increasing.
- In G', $c_\lambda(s, v)$ non-decreasing with decreasing λ.

Adapt Parametric Goldberg Tarjan to decreasing values of λ

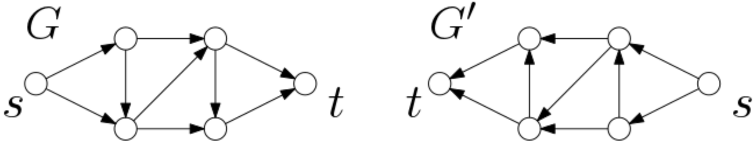
- **Idea:** From G obtain equivalent problem G' by reversing all arcs and interchanging source and sink.



- In G', $\lambda \mapsto c_\lambda(s, v)$ non-increasing.
- In G', $c_\lambda(s, v)$ non-decreasing with decreasing λ.
- Analogously: In G', $c_\lambda(v, t)$ non-increasing with decreasing λ.

Adapt Parametric Goldberg Tarjan to decreasing values of λ

- **Idea:** From G obtain equivalent problem G' by reversing all arcs and interchanging source and sink.



- In G', $\lambda \mapsto c_\lambda(s, v)$ non-increasing.
- In G', $c_\lambda(s, v)$ non-decreasing with decreasing λ.
- Analogously: In G', $c_\lambda(v, t)$ non-increasing with decreasing λ.
- Apply Parametric Goldberg Tarjan on G'.

Application: Flow sharing

Given: Directed Graph (V, E) with capacities $c \in \mathbb{R}^E$
Multiple sources $s_1, ..., s_k \in V$, sink $t \in V$
Source weights $w_1, ..., w_k > 0$

Application: Flow sharing

Given: Directed Graph (V, E) with capacities $c \in \mathbb{R}^E$
Multiple sources $s_1, ..., s_k \in V$, sink $t \in V$
Source weights $w_1, ..., w_k > 0$

Amount of flow that originates in source s_i:

$$u_i := \sum_{v \in V} f(s_i, v)$$

Application: Flow sharing

Given: Directed Graph (V, E) with capacities $c \in \mathbb{R}^E$
Multiple sources $s_1, ..., s_k \in V$, sink $t \in V$
Source weights $w_1, ..., w_k > 0$

Amount of flow that originates in source s_i:

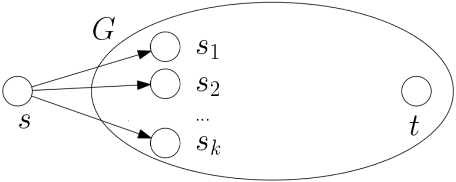
$$u_i := \sum_{v \in V} f(s_i, v)$$

Perfect sharing:

- Restrict to flows with equal ratios $\frac{u_1}{w_1}, ..., \frac{u_k}{w_k}$.
- Find maximal flow under this restriction.

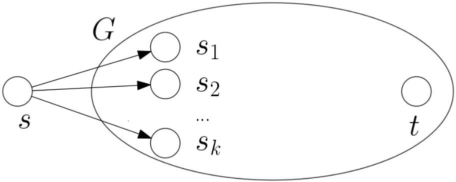
Solving perfect sharing:

- Add supersource s, edge (s, s_i) with $c_\lambda(s, s_i) = \lambda \cdot w_i$ for $i = 1, \ldots, k$.



Solving perfect sharing:

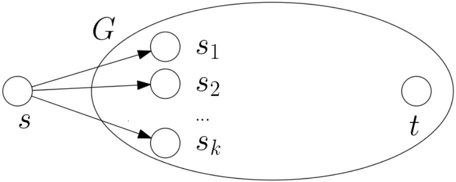
- Add supersource s, edge (s, s_i) with $c_\lambda(s, s_i) = \lambda \cdot w_i$ for $i = 1, ..., k$.



- Find smallest breakpoint λ_0 of $\kappa(\lambda)$ by similar algorithm.

Solving perfect sharing:

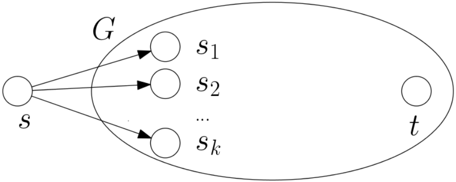
- Add supersource s, edge (s, s_i) with $c_\lambda(s, s_i) = \lambda \cdot w_i$ for $i = 1, ..., k$.



- Find smallest breakpoint λ_0 of $\kappa(\lambda)$ by similar algorithm.
- $\mathrm{MinCut} = (\{s\}, V)$ iff $\lambda \leq \lambda_0$.

Solving perfect sharing:

- Add supersource s, edge (s, s_i) with $c_\lambda(s, s_i) = \lambda \cdot w_i$ for $i = 1, ..., k$.

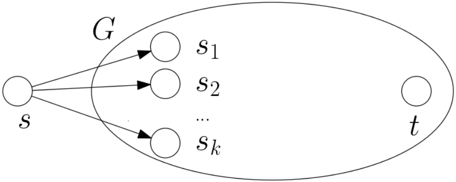


- Find smallest breakpoint λ_0 of $\kappa(\lambda)$ by similar algorithm.
- $\mathrm{MinCut} = (\{s\}, V)$ iff $\lambda \leq \lambda_0$.
- For $\lambda \leq \lambda_0$ have perfect sharing, since for all i:

$$\frac{u_i}{w_i} = \frac{f(s, s_i)}{w_i} = \frac{c_\lambda(s, s_i)}{w_i} = \lambda$$

Solving perfect sharing:

- Add supersource s, edge (s, s_i) with $c_\lambda(s, s_i) = \lambda \cdot w_i$ for $i = 1, ..., k$.



- Find smallest breakpoint λ_0 of $\kappa(\lambda)$ by similar algorithm.
- $\mathrm{MinCut} = (\{s\}, V)$ iff $\lambda \leq \lambda_0$.
- For $\lambda \leq \lambda_0$ have perfect sharing, since for all i:

$$\frac{u_i}{w_i} = \frac{f(s, s_i)}{w_i} = \frac{c_\lambda(s, s_i)}{w_i} = \lambda$$

- λ_0 maximizes MaxFlow under perfect sharing.

Other flow sharing problems

- MaxMin sharing: Among maximum flows, maximize $\min_{i=1,\dots,k} \frac{u_i}{w_i}$.

Other flow sharing problems

- MaxMin sharing: Among maximum flows, maximize $\min_{i=1,\dots,k} \frac{u_i}{w_i}$.
- MinMax sharing: Among maximum flows, minimize $\max_{i=1,\dots,k} \frac{u_i}{w_i}$.

Other flow sharing problems

- MaxMin sharing: Among maximum flows, maximize $\min_{i=1,\dots,k} \frac{u_i}{w_i}$.
- MinMax sharing: Among maximum flows, minimize $\max_{i=1,\dots,k} \frac{u_i}{w_i}$.
- Optimal sharing: Among maximum flows, maximize $\min_{i=1,\dots,k} \frac{u_i}{w_i}$ and minimize $\max_{i=1,\dots,k} \frac{u_i}{w_i}$ simultaneously.

Other flow sharing problems

- MaxMin sharing: Among maximum flows, maximize $\min_{i=1,\ldots,k} \frac{u_i}{w_i}$.
- MinMax sharing: Among maximum flows, minimize $\max_{i=1,\ldots,k} \frac{u_i}{w_i}$.
- Optimal sharing: Among maximum flows,
  maximize $\min_{i=1,\ldots,k} \frac{u_i}{w_i}$ and minimize $\max_{i=1,\ldots,k} \frac{u_i}{w_i}$ simultaneously.
- Lexicographic sharing: Among maximum flows, minimize

  $$\frac{u_{i_1}}{w_{i_1}} < \ldots < \frac{u_{i_1}}{w_{i_1}}$$

  lexicograpically.

Other flow sharing problems

- MaxMin sharing: Among maximum flows, maximize $\min_{i=1,\dots,k} \frac{u_i}{w_i}$.
- MinMax sharing: Among maximum flows, minimize $\max_{i=1,\dots,k} \frac{u_i}{w_i}$.
- Optimal sharing: Among maximum flows, maximize $\min_{i=1,\dots,k} \frac{u_i}{w_i}$ and minimize $\max_{i=1,\dots,k} \frac{u_i}{w_i}$ simultaneously.
- Lexicographic sharing: Among maximum flows, minimize

$$\frac{u_{i_1}}{w_{i_1}} < \dots < \frac{u_{i_1}}{w_{i_1}}$$

lexicograpically.

Parametric Goldberg Tarjan can solve all of them!

References

- Gallo, Grigoriadis & Tarjan (1989). *A fast parametric maximum flow algorithm and applications.* SIAM J. Comput. 18 (1), pp. 30