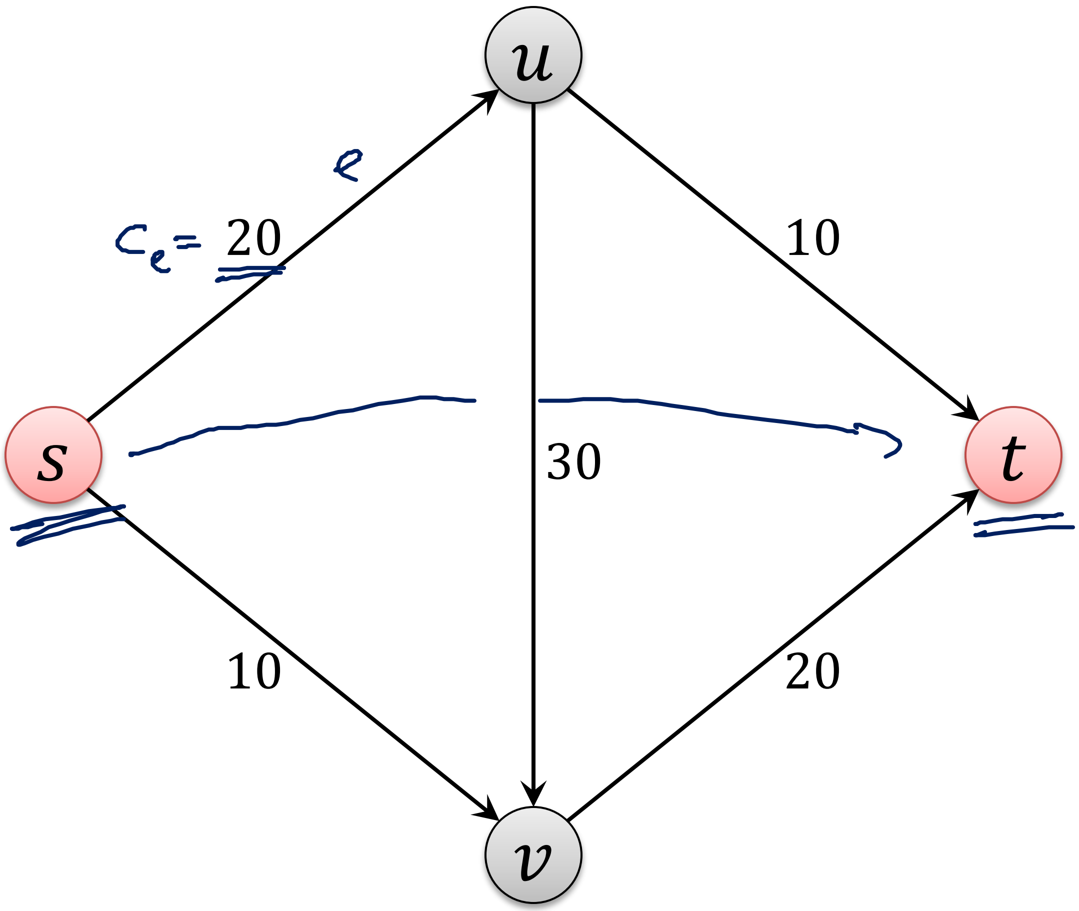
Chapter 6
Graph Algorithms

Algorithm Theory
WS 2015/16

Fabian Kuhn

Example: Flow Network



Notation

We define:

$$f^{\text{in}}(v) := \sum_{e \text{ into } v} f(e), \qquad f^{\text{out}}(v) := \sum_{e \text{ out of } v} f(e)$$

$0 \leq f(e) \leq c_e$

For a set $S \subseteq V$:

$$f^{\text{in}}(S) := \sum_{e \text{ into } S} f(e), \qquad f^{\text{out}}(S) := \sum_{e \text{ out of } S} f(e)$$

Flow conservation: $\forall v \in V \setminus \{s, t\}$: $f^{\text{in}}(v) = f^{\text{out}}(v)$

Flow value: $|f| = f^{\text{out}}(s) = f^{\text{in}}(t)$

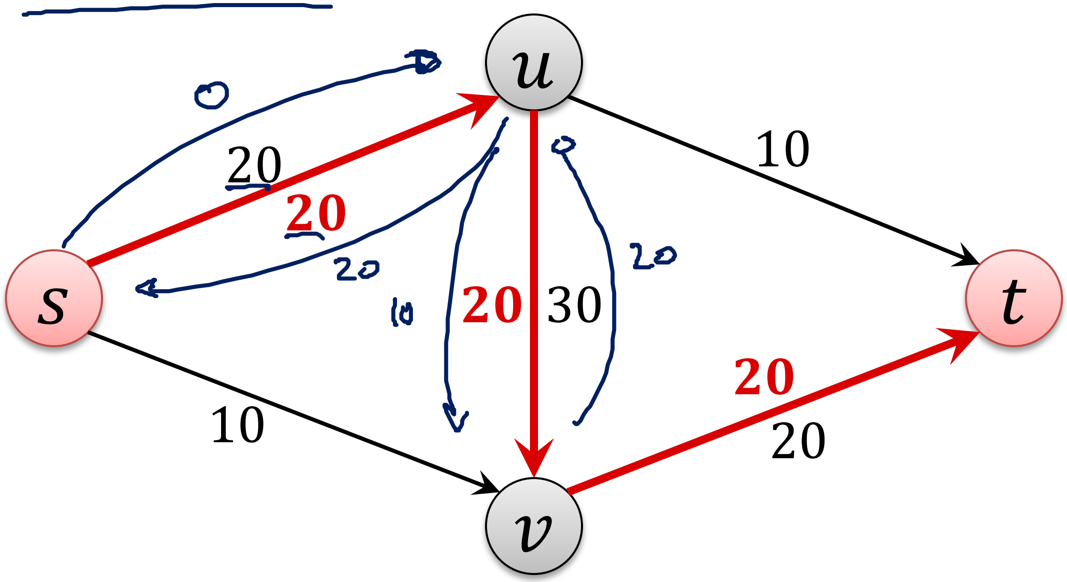
For simplicity: Assume that all capacities are positive integers

Residual Graph

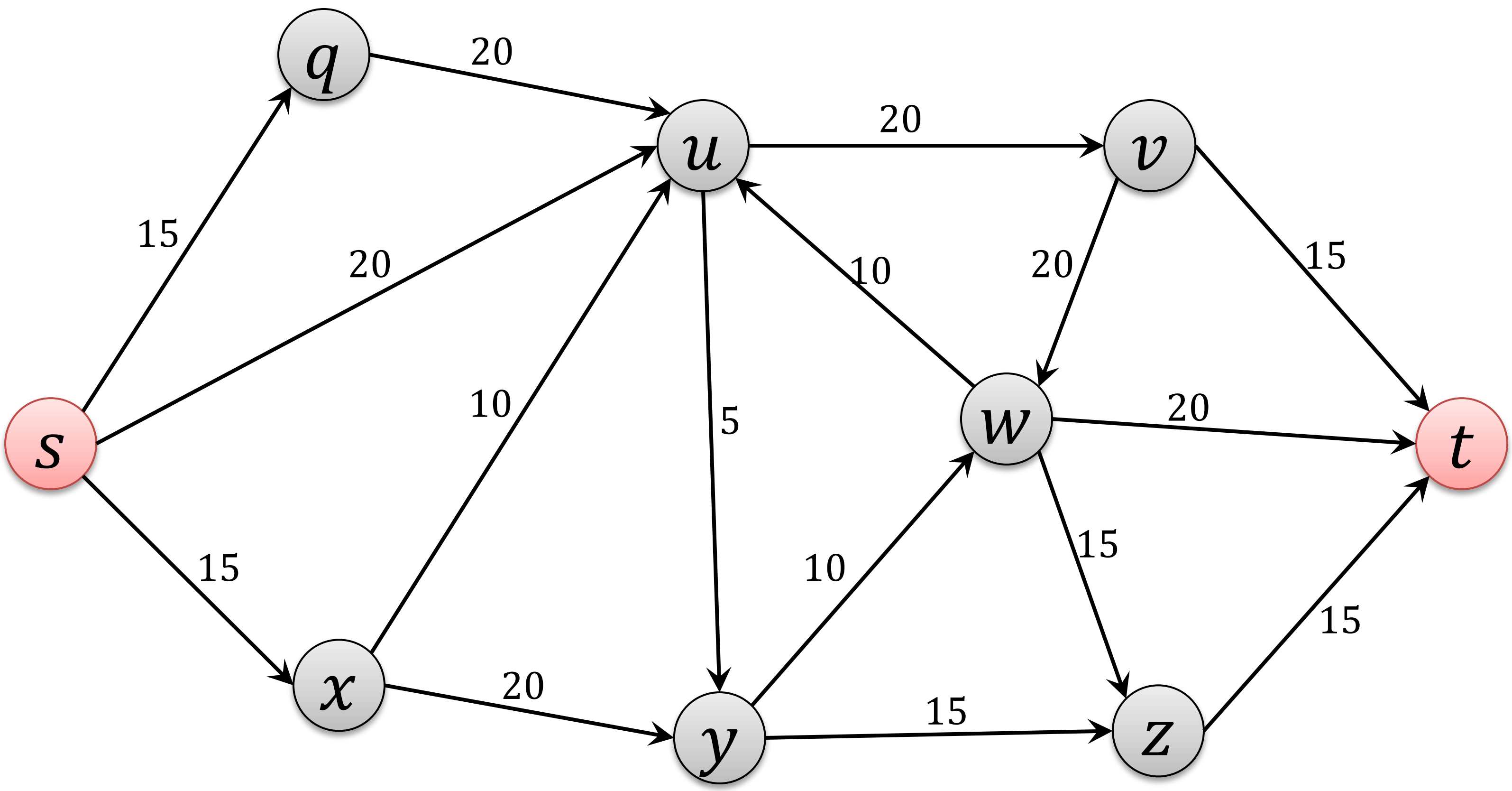
Given a flow network $G = (V, E)$ with capacities c_e (for $e \in E$)

For a flow f on G, define directed graph $G_f = (V_f, E_f)$ as follows:

- Node set $V_f = V$
- For each edge $e = (u, v)$ in E, there are two edges in E_f:
  - forward edge $e = (u, v)$ with residual capacity $c_e - f(e)$
  - backward edge $e' = (v, u)$ with residual capacity $f(e)$

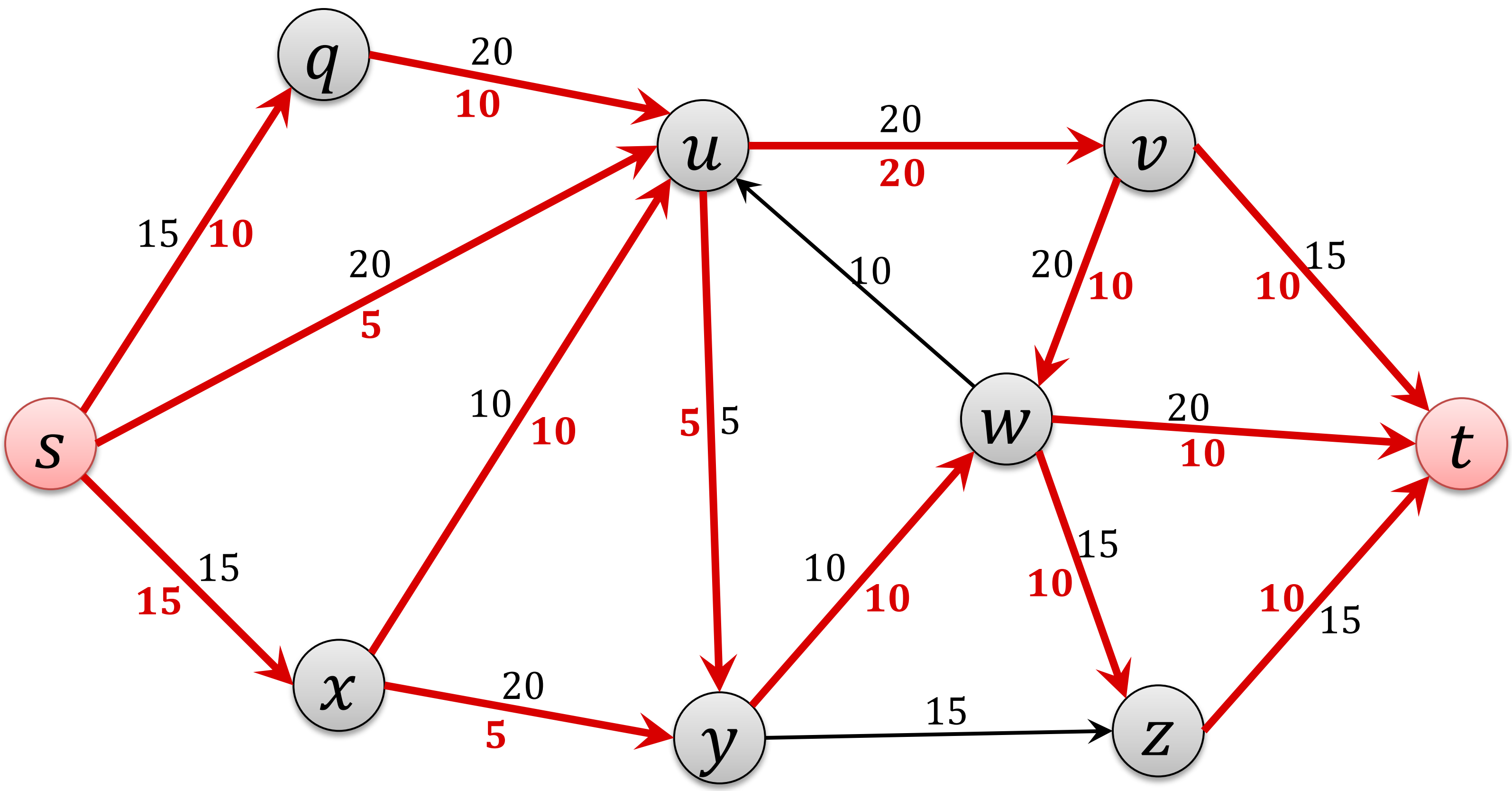


Residual Graph: Example



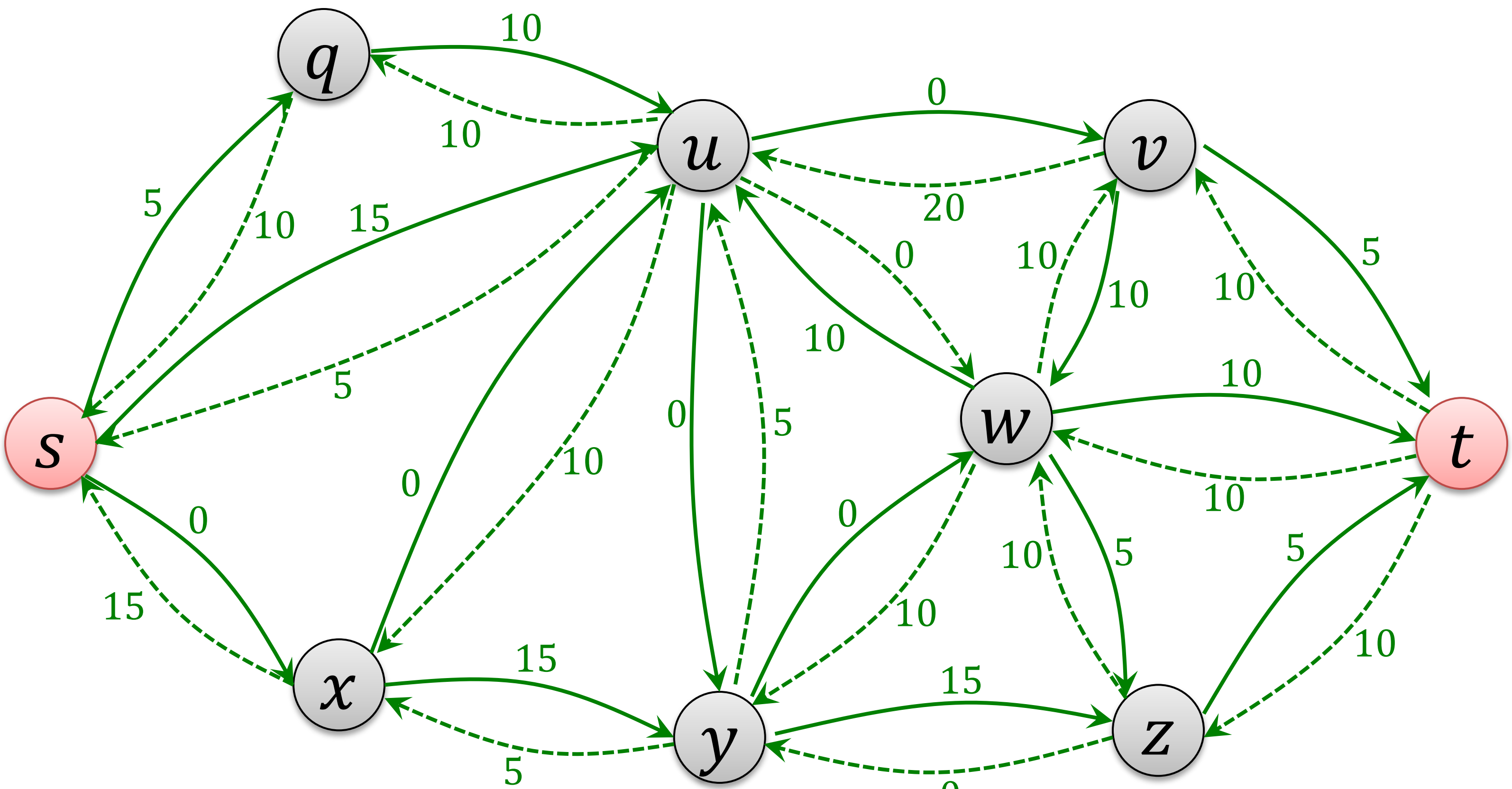
Residual Graph: Example

Flow f



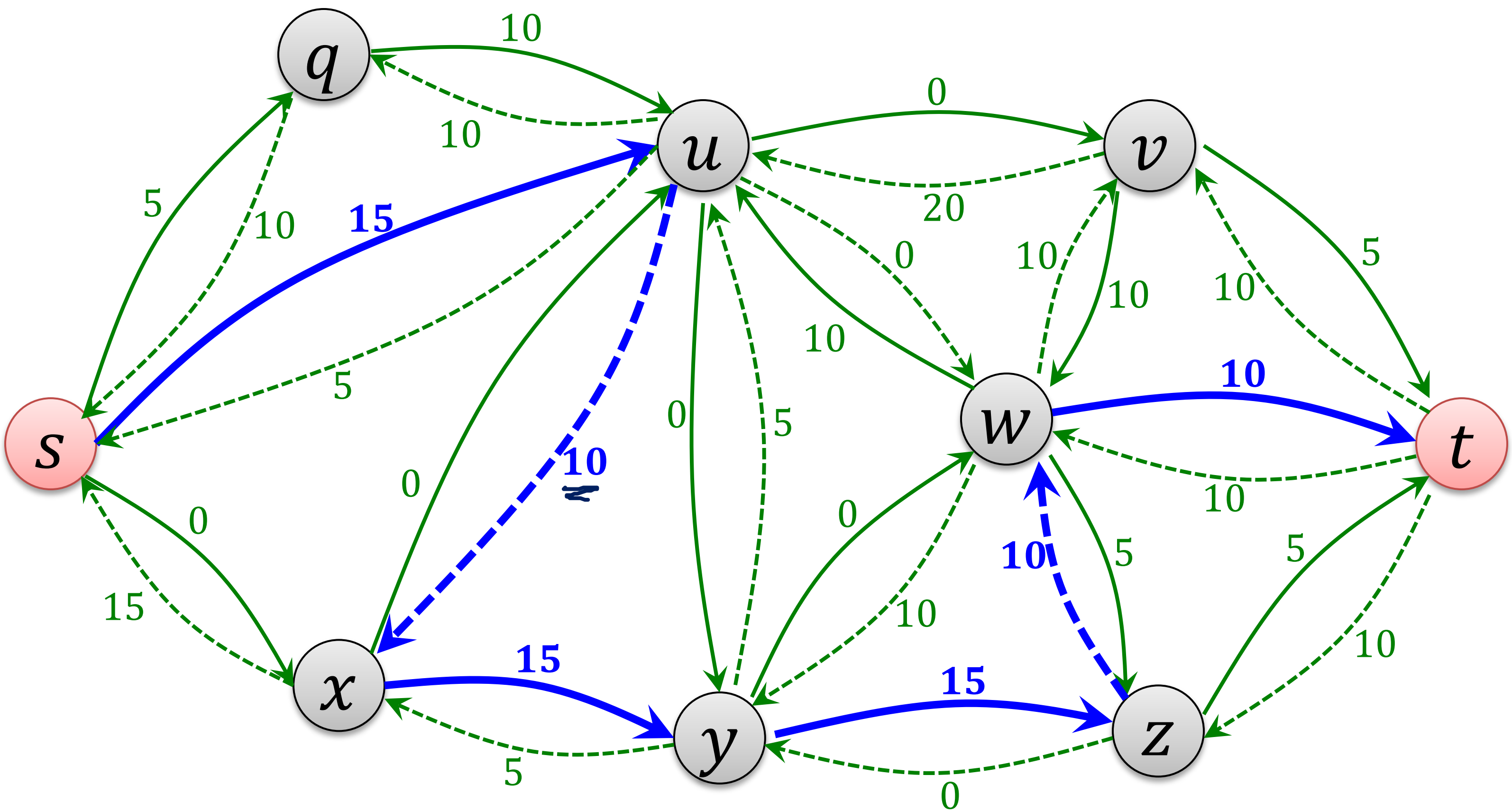
Residual Graph: Example

Residual Graph G_f



Augmenting Path

Residual Graph G_f



Augmenting Path

Definition:
An augmenting path P is a (simple) s-t-path on the residual graph G_f on which each edge has residual capacity > 0.

$\text{bottleneck}(P, f)$: minimum residual capacity on any edge of the augmenting path P

Augment flow f to get flow f':

- For every forward edge (u, v) on P:

$$\boldsymbol{f'\big((u,v)\big) := f\big((u,v)\big) + \textbf{bottleneck}(P,f)}$$

- For every backward edge (u, v) on P:

$$\boldsymbol{f'\big((v,u)\big) := f\big((v,u)\big) - \textbf{bottleneck}(P,f)}$$

Augmented Flow

Lemma: Given a flow f and an augmenting path P, the resulting augmented flow f' is legal and its value is

$$|f'| = |f| + \textbf{bottleneck}(P, f).$$

Proof:

Ford-Fulkerson Algorithm

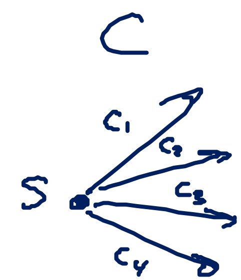
- Improve flow using an augmenting path as long as possible:

1. Initially, $f(e) = 0$ for all edges $e \in E$, $G_f = G$
2. **while** there is an augmenting s-t-path P in G_f **do**
3. Let P be an augmenting s-t-path in G_f;
4. $f' \coloneqq \mathrm{augment}(f, P)$; bottleneck$(P, f) > 0$
5. update f to be f';
6. update the residual graph $G_{f'}$
7. **end**;

Ford-Fulkerson Running Time

Theorem: If all edge capacities are integers, the Ford-Fulkerson algorithm terminates after at most C iterations, where

$$C = \sum_{e \text{ out of } s} c_e .$$



Proof:

At all times, for each $e \in E$: $f(e)$ is an integer

initially: $f(e) = 0$

in one iter.: augm. path P : residual cap. are integers

bottleneck$(P, f) > 0$ (it also is an int.)

$\to \geq 1$

$\to$ new flow values are integers

$\to$ new flow value larger by ≥ 1

every flow value $\leq C$

Ford-Fulkerson Running Time

Theorem: If all edge capacities are integers, the Ford-Fulkerson algorithm can be implemented to run in $O(mC)$ time.

#edges

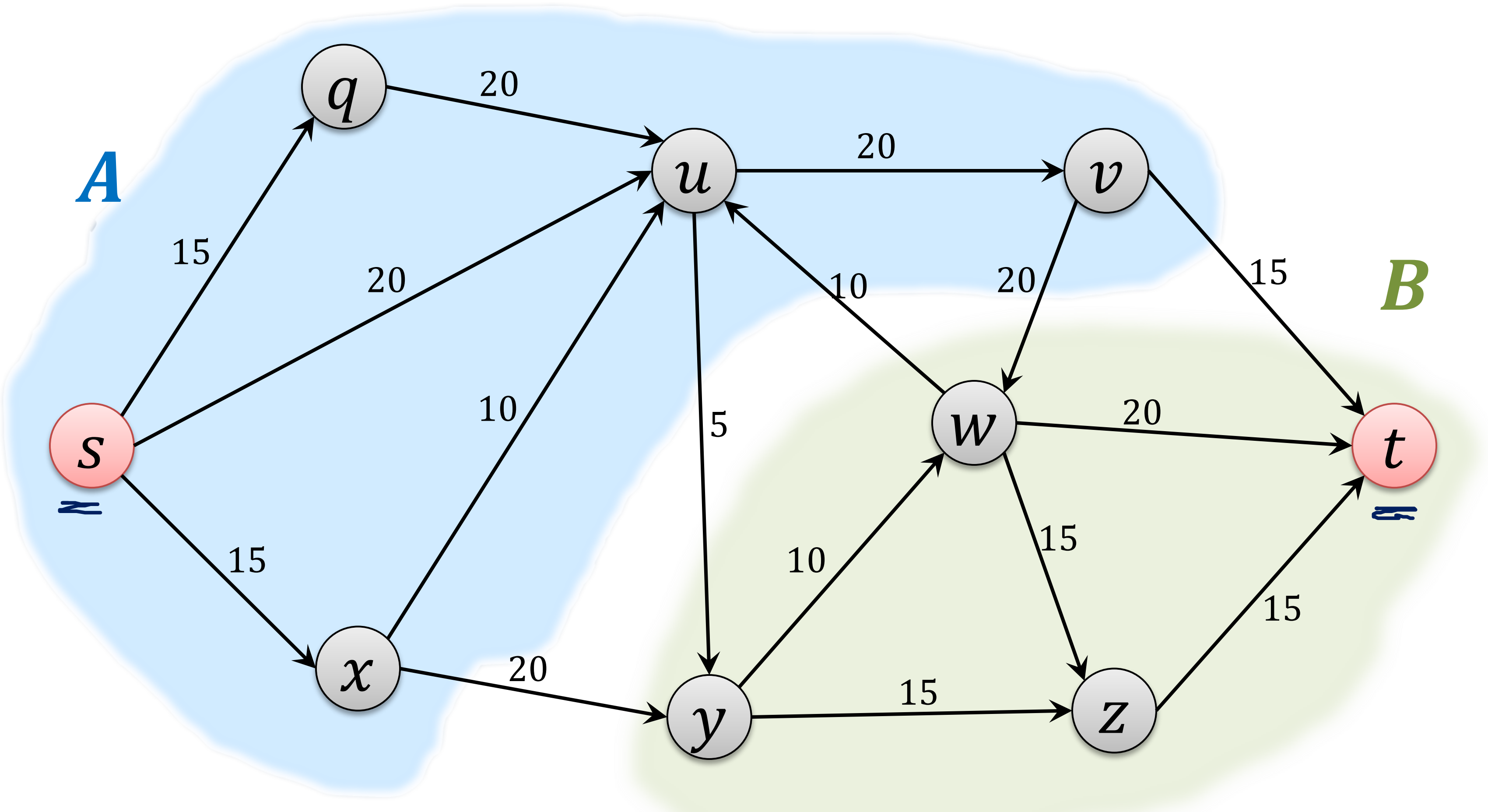
Proof:

Claim: one iter. can be computed in $O(m)$ time

1. compute/update residual graph G_f
   - first iter: $O(m)$
   - later iter: $O(n)$
2. find augm. path / conclude there is no augm. path
   - s-t path in G_f with res. cap. > 0
   - graph traversal (DFS/BFS): $O(m)$ time
3. update flow values: $O(n)$ time

s-t Cuts

Definition:
An s-t cut is a partition (A, B) of the vertex set such that $s \in A$ and $t \in B$

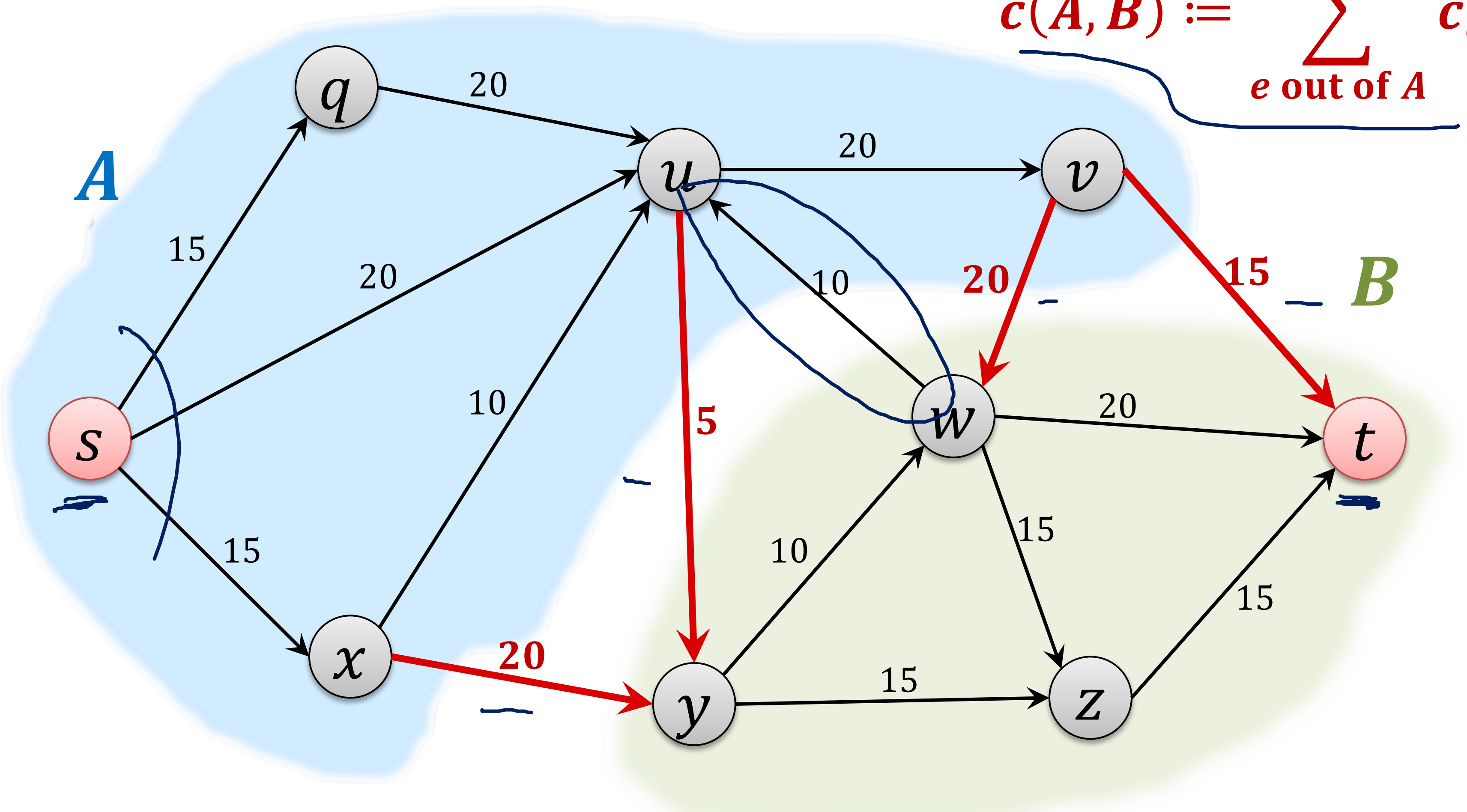


Cut Capacity

Definition:

The capacity $c(A, B)$ of an s-t-cut (A, B) is defined as

$$c(A, B) := \sum_{e \text{ out of } A} c_e .$$

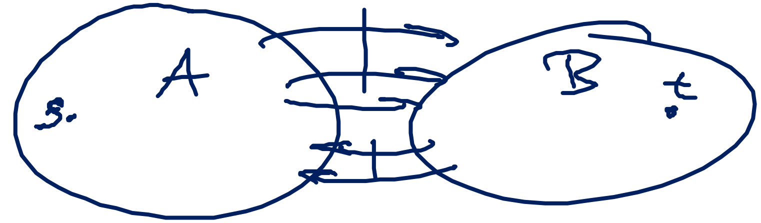


Cuts and Flow Value

Lemma: Let f be any s-t flow, and (A, B) any s-t cut. Then,

$$|\boldsymbol{f}| = \boldsymbol{f}^{\mathbf{out}}(\boldsymbol{A}) - \boldsymbol{f}^{\mathbf{in}}(\boldsymbol{A}).$$

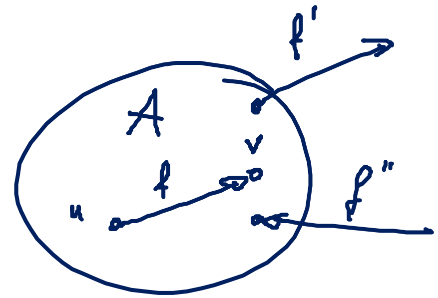
Proof:



$|f| = f^{out}(s) \quad (= f^{in}(t))$

$|f| = f^{out}(s) - \underbrace{f^{in}(s)}_{=0}$

$= \sum_{v \in A} \underbrace{\left(f^{out}(v) - f^{in}(v)\right)}_{=0 \text{ except for } v=s} \quad \left(\forall v \in A \setminus \{s\}: f^{out}(v) = f^{in}(v)\right)$



$= f^{out}(A) - f^{in}(A)$

Cuts and Flow Value

Lemma: Let f be any s-t flow, and (A, B) any s-t cut. Then,

$$|f| = f^{\text{out}}(A) - f^{\text{in}}(A).$$

Lemma: Let f be any s-t flow, and (A, B) any s-t cut. Then,

$$|f| = f^{\text{in}}(B) - f^{\text{out}}(B).$$

Proof:

symmetric

or observe

$$f^{\text{out}}(A) = f^{\text{in}}(B)$$

$$f^{\text{in}}(A) = f^{\text{out}}(B)$$

Upper Bound on Flow Value

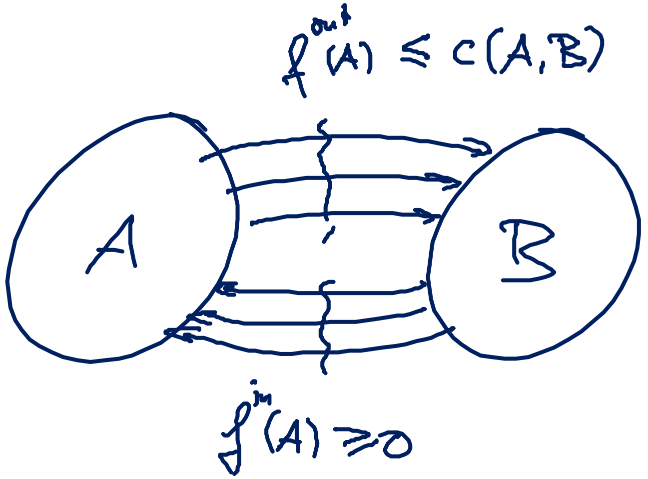
Lemma:

Let f be any s-t flow and (A, B) any s-t cut. Then $|\boldsymbol{f}| \leq \boldsymbol{c}(\boldsymbol{A}, \boldsymbol{B})$.

Proof:

$$|f| = f^{out}(A) - f^{in}(A)$$

$$\leq c(A,B) - 0$$



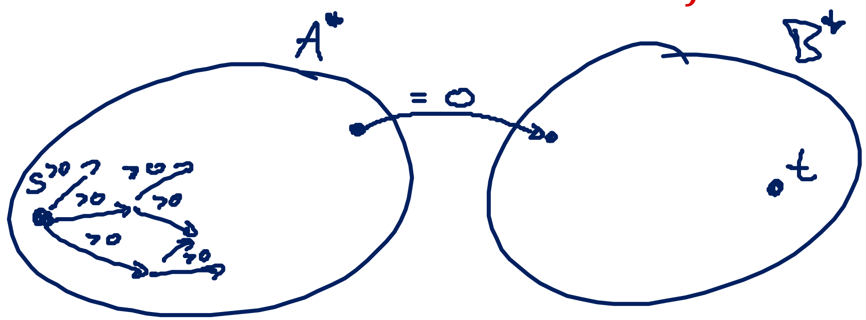
Ford-Fulkerson Gives Optimal Solution

Lemma: If f is an s-t flow such that there is no augmenting path in G_f, then there is an s-t cut (A^*, B^*) in G for which

$$|\boldsymbol{f}| = \boldsymbol{c}(\boldsymbol{A}^*, \boldsymbol{B}^*).$$

Proof:

- Define $\boldsymbol{A}^*$: set of nodes that can be reached from s on a path with positive residual capacities in G_f:



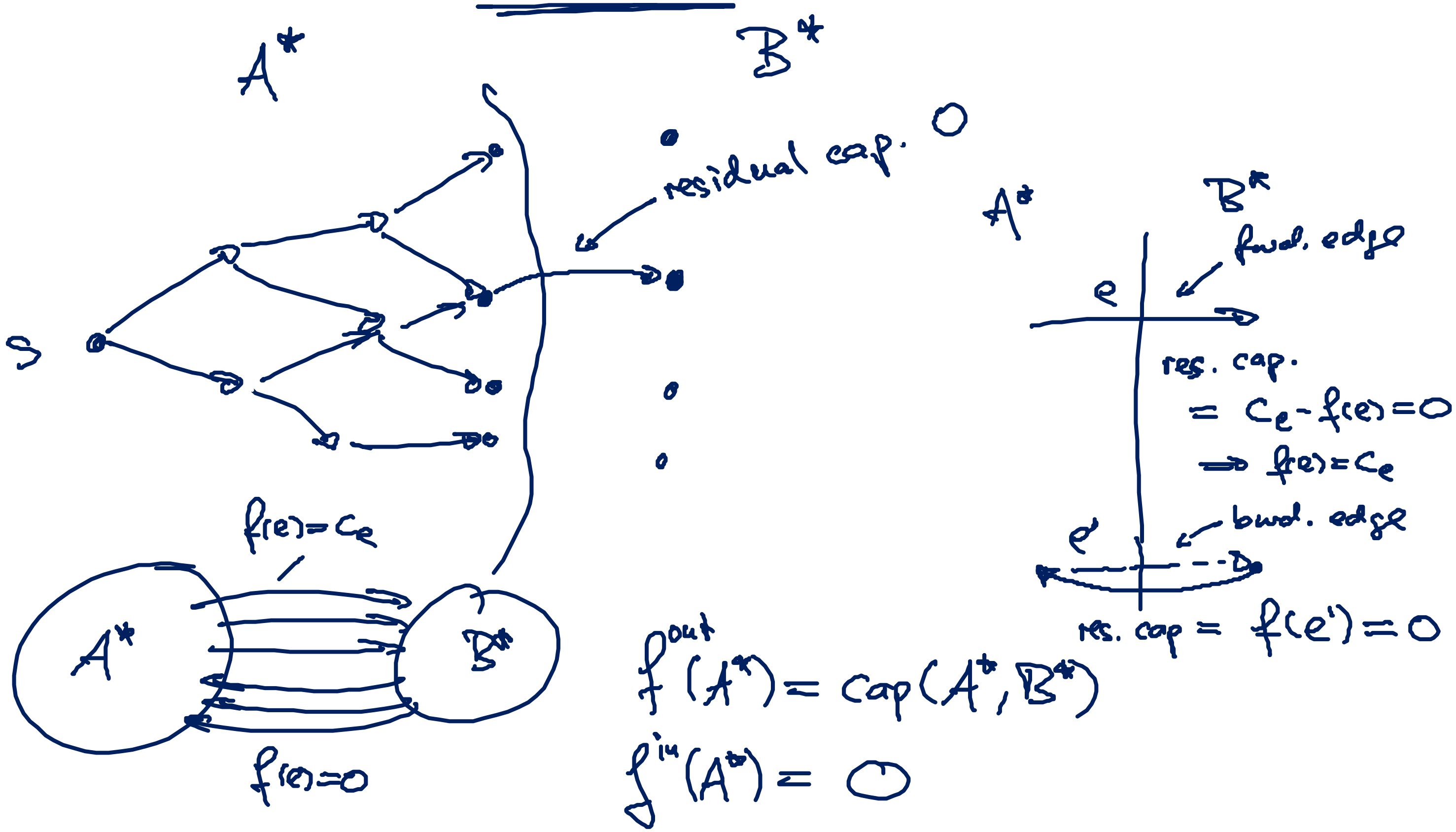
- For $B^* = V \setminus A^*$, (A^*, B^*) is an s-t cut
  - By definition $s \in A^*$ and $t \notin A^*$ ← because there is no augm. path

Ford-Fulkerson Gives Optimal Solution

Lemma: If f is an s-t flow such that there is no augmenting path in G_f, then there is an s-t cut (A^*, B^*) in G for which

$$|f| = c(A^*, B^*).$$

Proof:



Ford-Fulkerson Gives Optimal Solution

Lemma: If f is an s-t flow such that there is no augmenting path in G_f, then there is an s-t cut (A^*, B^*) in G for which

$$|\boldsymbol{f}| = \boldsymbol{c}(\boldsymbol{A}^*, \boldsymbol{B}^*).$$

Proof:

Ford-Fulkerson Gives Optimal Solution

Theorem: The flow returned by the Ford-Fulkerson algorithm is a maximum flow.

Proof:

f^*: flow returned by FF

↳ cut (A^*, B^*)

s.t. $|f^*| = c(A^*, B^*)$

for every flow f: $|f| \leq c(A^*, B^*)$

Min-Cut Algorithm

Ford-Fulkerson also gives a min-cut algorithm:

Theorem: Given a flow f of maximum value, we can compute an s-t cut of minimum capacity in $O(m)$ time.

Proof:

f maximum $\longrightarrow$ augm. path

can find cut (A^*, B^*) s.t. $|f| = c(A^*, B^*)$

↳ as before: DFS / BFS on res. graph (from s)

↳ all nodes reachable from s

↳ A^* (set of nodes reachable from s)

$\longrightarrow$ A^* can be computed in $O(m)$ time

(A^*, B^*) is an s-t cut with min. capacity

because: for every other s-t cut (A, B), we have $|f| \leq c(A, B)$

$$|f| = c(A^*, B^*) \leq c(A, B)$$

Max-Flow Min-Cut Theorem

Theorem: **(Max-Flow Min-Cut Theorem)**

In every flow network, the maximum value of an s-t flow is equal to the minimum capacity of an s-t cut.

Proof:

FF gives max flow f^* and min s-t cut (A^*, B^*)

s.t. $|f^*| = c(A^*, B^*)$

Integer Capacities

Theorem: **(Integer-Valued Flows)**

If all capacities in the flow network are integers, then there is a maximum flow f for which the flow $f(e)$ of every edge e is an integer.

Proof:

FF gives an integer flow

Non-Integer Capacities

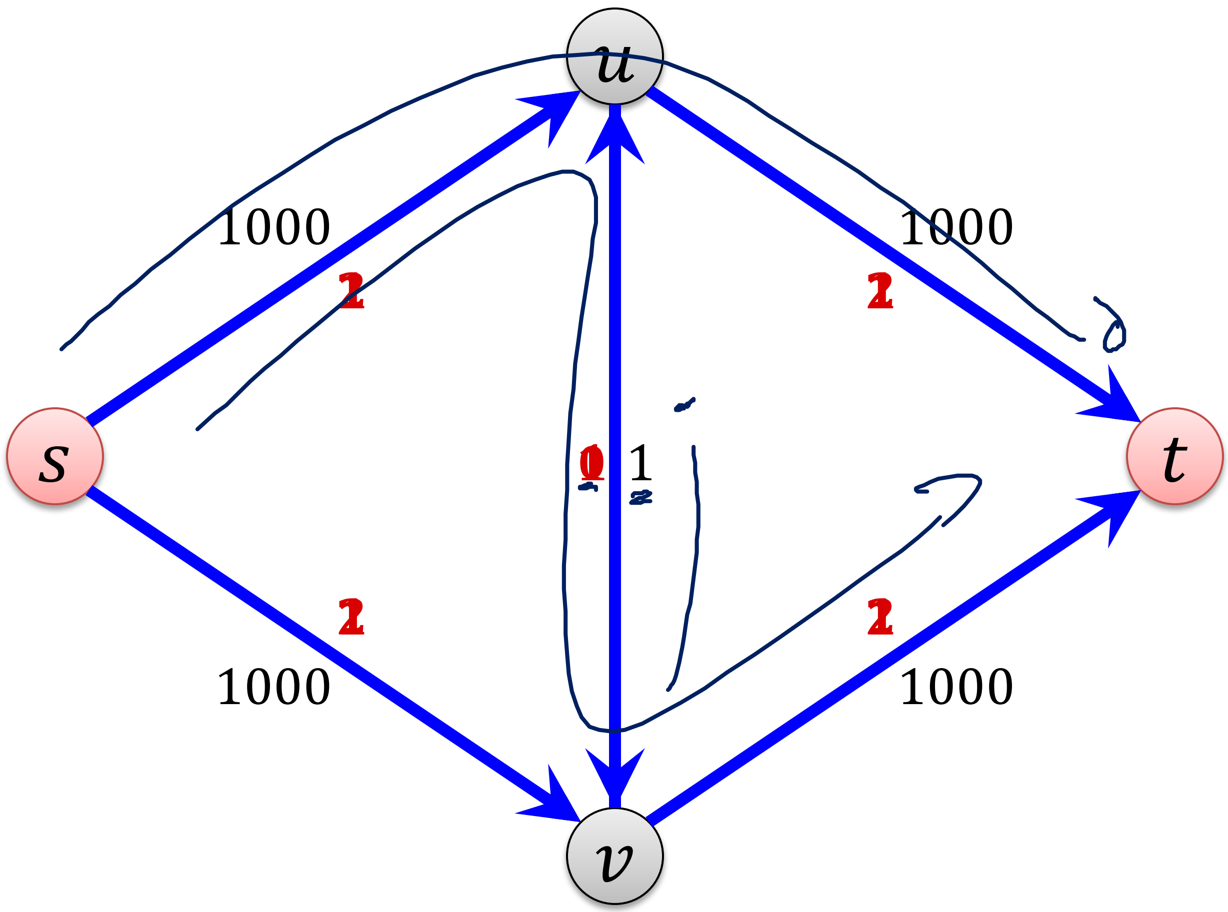
What if capacities are not integers?

$O(m\,C)$

- rational capacities: $c_e \in \mathbb{Q}$
  - can be turned into integers by multiplying them with large enough integer
  - algorithm still works correctly
- real (non-rational) capacities:
  - not clear whether the algorithm always terminates
- even for integer capacities, time can linearly depend on the value of the maximum flow

$C \longrightarrow \log C$

Slow Execution



- Number of iterations: 2000 (value of max. flow)

Improved Algorithm

Idea: Find the best augmenting path in each step

- best: path P with maximum $\text{bottleneck}(P, f)$
- Best path might be rather expensive to find
  → find almost best path

Δ always a power of 2

- **Scaling parameter Δ:**
  (initially, $\Delta =$ "$\max c_e$ rounded down to next power of 2")
- As long as there is an augmenting path that improves the flow by at least Δ, augment using such a path
- If there is no such path: $\Delta := \Delta/2$

Scaling Parameter Analysis

Lemma: If all capacities are integers, number of different scaling parameters used is $\leq 1 + \lfloor \log_2 C \rfloor$.

C_{max}: max. edge cap.

for all e: $c_e \leq C$

initially

$\Delta = 2^{\lfloor \log_2 C_{max} \rfloor}$

largest Δ

\# of scaling param: $\leq \lfloor \log_2 C_{max} \rfloor + 1$

- **Δ-scaling phase:** Time during which scaling parameter is Δ

running time:

\#phases · #iter. per phase · $O(m)$

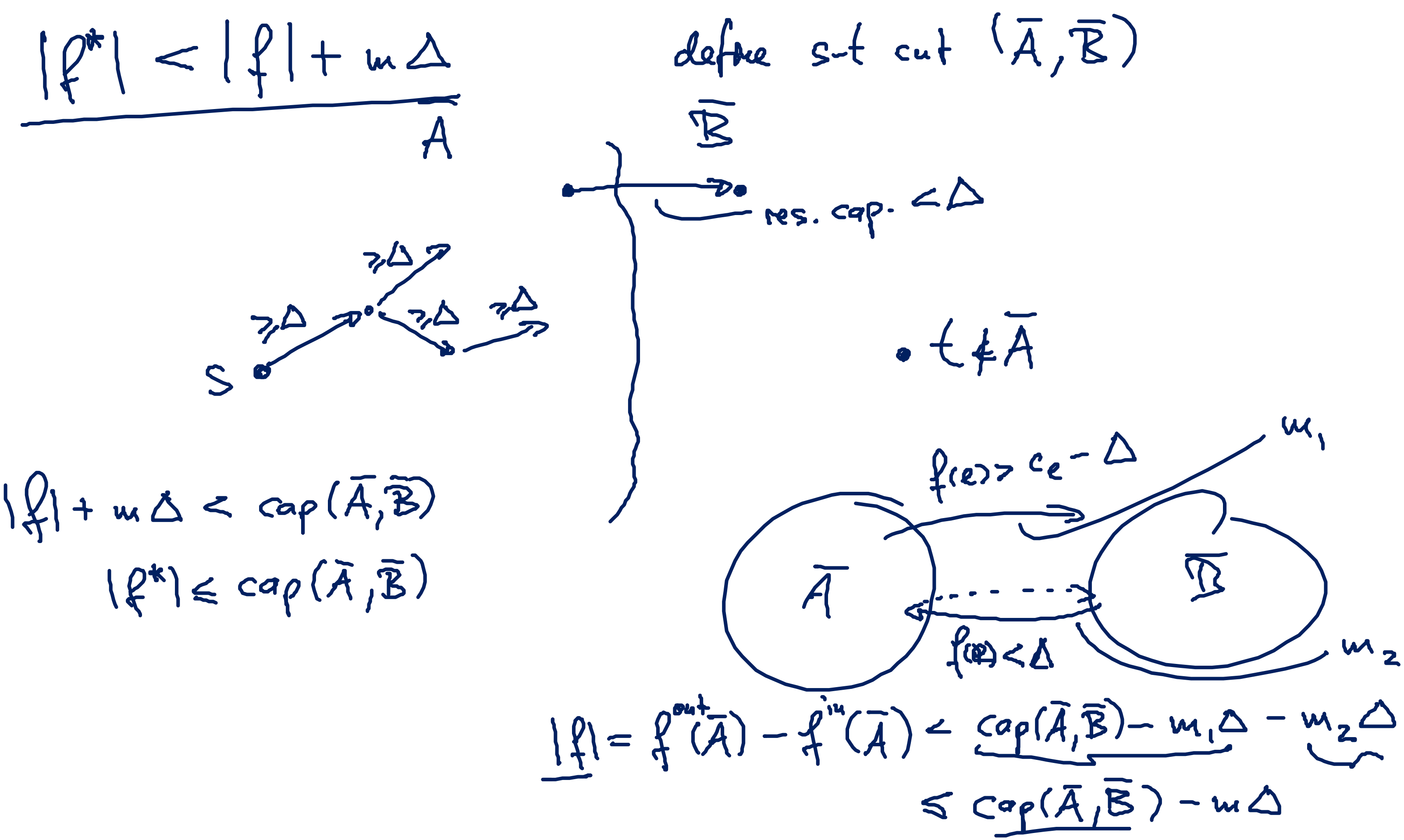
\#phases: $O(\log C)$

\#iter. per phase: ?

$O(m)$: find one path

Length of a Scaling Phase

Lemma: If f is the flow at the end of the Δ-scaling phase, the maximum flow in the network has value at most $|f| + m\Delta$.



Length of a Scaling Phase

Lemma: The number of augmentation in each scaling phase is at most $2m$.

at the beginning of the Δ-scaling phase

↳ at the end of the 2Δ-scaling phase

$\Longrightarrow |f^*| < |f| + 2m\Delta$ (prev. lemma)

each augm. path improves $|f|$ by $\geq \Delta$

$\square$

running time: $O(\log C) \cdot O(m) \cdot O(m) = O(m^2 \log C)$

Running Time: Scaling Max Flow Alg.

Theorem: The number of augmentations of the algorithm with scaling parameter and integer capacities is at most $O(m \log C)$. The algorithm can be implemented in time $O(m^2 \log C)$.

Strongly Polynomial Algorithm

- Time of regular Ford-Fulkerson algorithm with integer capacities:

$$O(mC)$$

- Time of algorithm with scaling parameter:

$$O(m^2 \log C)$$

- $O(\log C)$ is polynomial in the size of the input, but not in n
- Can we get an algorithm that runs in time polynomial in n?
- Always picking a shortest augmenting path leads to running time

$$O(m^2 n)$$

works if cap. are reals

Other Algorithms

- There are many other algorithms to solve the maximum flow problem, for example:

- **Preflow-push algorithm:**
  - Maintains a preflow (∀ nodes: inflow ≥ outflow)
  - Alg. guarantees: As soon as we have a flow, it is optimal
  - Detailed discussion in ~~last year's~~ 2012/13 lecture
  - Running time of basic algorithm: $O(m \cdot n^2)$
  - Doing steps in the "right" order: $O(n^3)$

- **Current best known complexity:** $\boldsymbol{O(m \cdot n)}$
  - For graphs with $m \geq n^{1+\epsilon}$ [King,Rao,Tarjan 1992/1994]
    (for every constant $\epsilon > 0$)
  - For sparse graphs with $m \leq n^{16/15-\delta}$ [Orlin, 2013]

max. flow in undirected networks (1+ε)-approx. max flow. $O(m \cdot n^{o(1)})$ necessary

Maximum Flow Applications

- Maximum flow has many applications

- Reducing a problem to a max flow problem can even be seen as an important algorithmic technique

- Examples:
  - related network flow problems
  - computation of small cuts
  - computation of matchings
  - computing disjoint paths
  - scheduling problems
  - assignment problems with some side constraints
  - …

Undirected Edges and Vertex Capacities

Undirected Edges:

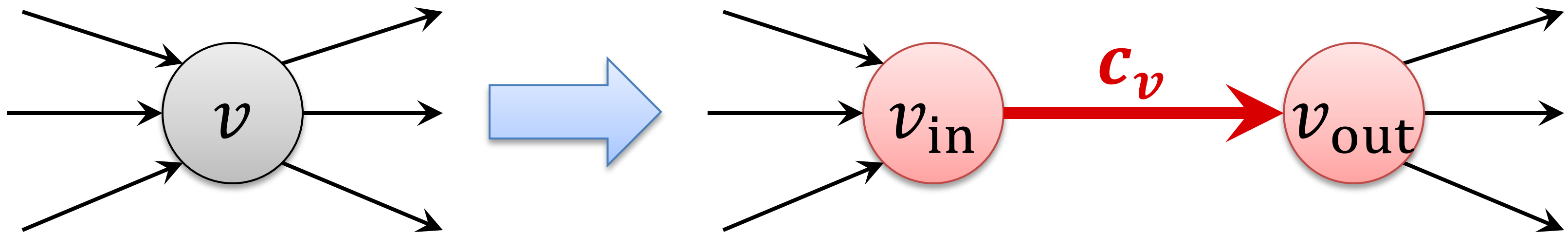
- Undirected edge $\{u, v\}$: add edges (u, v) and (v, u) to network

Vertex Capacities:

- Not only edges, but also (or only) nodes have capacities
- Capacity c_v of node $v \notin \{s, t\}$:

$$f^{\text{in}}(v) = f^{\text{out}}(v) \leq c_v$$

- Replace node v by edge $e_v = \{v_{\text{in}}, v_{\text{out}}\}$:



Minimum s-t Cut

Given: undirected graph $G = (V, E)$, nodes $s, t \in V$

s-t cut: Partition (A, B) of V such that $s \in A, t \in B$

Size of cut (A, B): number of edges crossing the cut

Objective: find s-t cut of minimum size

Edge Connectivity

Definition: A graph $G = (V, E)$ is k-edge connected for an integer $k \geq 1$ if the graph $G_X = (V, E \setminus X)$ is connected for every edge set

$$X \subseteq E, |X| \leq k - 1.$$

Goal: Compute edge connectivity $\lambda(G)$ of G
(and edge set X of size $\lambda(G)$ that divides G into ≥ 2 parts)

- minimum set X is a minimum s-t cut for some $s, t \in V$
  - Actually for all s, t in different components of $G_X = (V, E \setminus X)$
- Possible algorithm: fix s and find min s-t cut for all $t \neq s$

Minimum s-t Vertex-Cut

Given: undirected graph $G = (V, E)$, nodes $s, t \in V$

***s*-*t* vertex cut:** Set $X \subset V$ such that $s, t \notin X$ and s and t are in different components of the sub-graph $G[V \setminus X]$ induced by $V \setminus X$

Size of vertex cut: $|X|$

Objective: find s-t vertex-cut of minimum size

- Replace undirected edge $\{u, v\}$ by (u, v) and (v, u)
- Compute max s-t flow for edge capacities ∞ and node capacities

$$c_v = 1 \text{ for } v \neq s, t$$

- Replace each node v by v_{in} and v_{out}:
- Min edge cut corresponds to min vertex cut in G

Vertex Connectivity

Definition: A graph $G = (V, E)$ is k-vertex connected for an integer $k \geq 1$ if the sub-graph $G[V \setminus X]$ induced by $V \setminus X$ is connected for every edge set

$$X \subseteq V, |X| \leq k - 1.$$

Goal: Compute vertex connectivity $\kappa(G)$ of G

(and node set X of size $\kappa(G)$ that divides G into ≥ 2 parts)

- Compute minimum s-t vertex cut for fixed s and all $t \neq s$

Edge-Disjoint Paths

Given: Graph $G = (V, E)$ with nodes $s, t \in V$

Goal: Find as many edge-disjoint s-t paths as possible

Solution:

- Find max s-t flow in G with edge capacities $c_e = 1$ for all $e \in E$

Flow f induces $|f|$ edge-disjoint paths:

- Integral capacities → can compute integral max flow f
- Get $|f|$ edge-disjoint paths by greedily picking them
- Correctness follows from flow conservation $f^{\text{in}}(v) = f^{\text{out}}(v)$

Vertex-Disjoint Paths

Given: Graph $G = (V, E)$ with nodes $s, t \in V$

Goal: Find as many internally vertex-disjoint s-t paths as possible

Solution:

- Find max s-t flow in G with node capacities $c_v = 1$ for all $v \in V$

Flow f induces $|f|$ vertex-disjoint paths:

- Integral capacities → can compute integral max flow f
- Get $|f|$ vertex-disjoint paths by greedily picking them
- Correctness follows from flow conservation $f^{\text{in}}(v) = f^{\text{out}}(v)$

Menger’s Theorem

Theorem: (edge version)
For every graph $G = (V, E)$ with nodes $s, t \in V$, the size of the minimum s-t (edge) cut equals the maximum number of pairwise edge-disjoint paths from s to t.

Theorem: (node version)
For every graph $G = (V, E)$ with nodes $s, t \in V$, the size of the minimum s-t vertex cut equals the maximum number of pairwise internally vertex-disjoint paths from s to t

- Both versions can be seen as a special case of the max flow min cut theorem

Baseball Elimination

Team i	Wins w_i	Losses ℓ_i	To Play r_i	Against = r_{ij}				
				NY	Balt.	T. Bay	Tor.	Bost.
New York	81	70	11	-	2	4	2	3
Baltimore	79	77	6	2	-	2	1	1
Tampa Bay	79	75	8	4	2	-	1	1
Toronto	76	80	6	2	1	1	-	2
Boston	71	84	7	3	1	1	2	-

- Only wins/losses possible (no ties), winner: team with most wins
- Which teams can still win (as least as many wins as top team)?
- Boston is eliminated (cannot win):
  - Boston can get at most 78 wins, New York already has 81 wins
- If for some i, j: $w_i + r_i < w_j$ → team i is eliminated
- Sufficient condition, but not a necessary one!

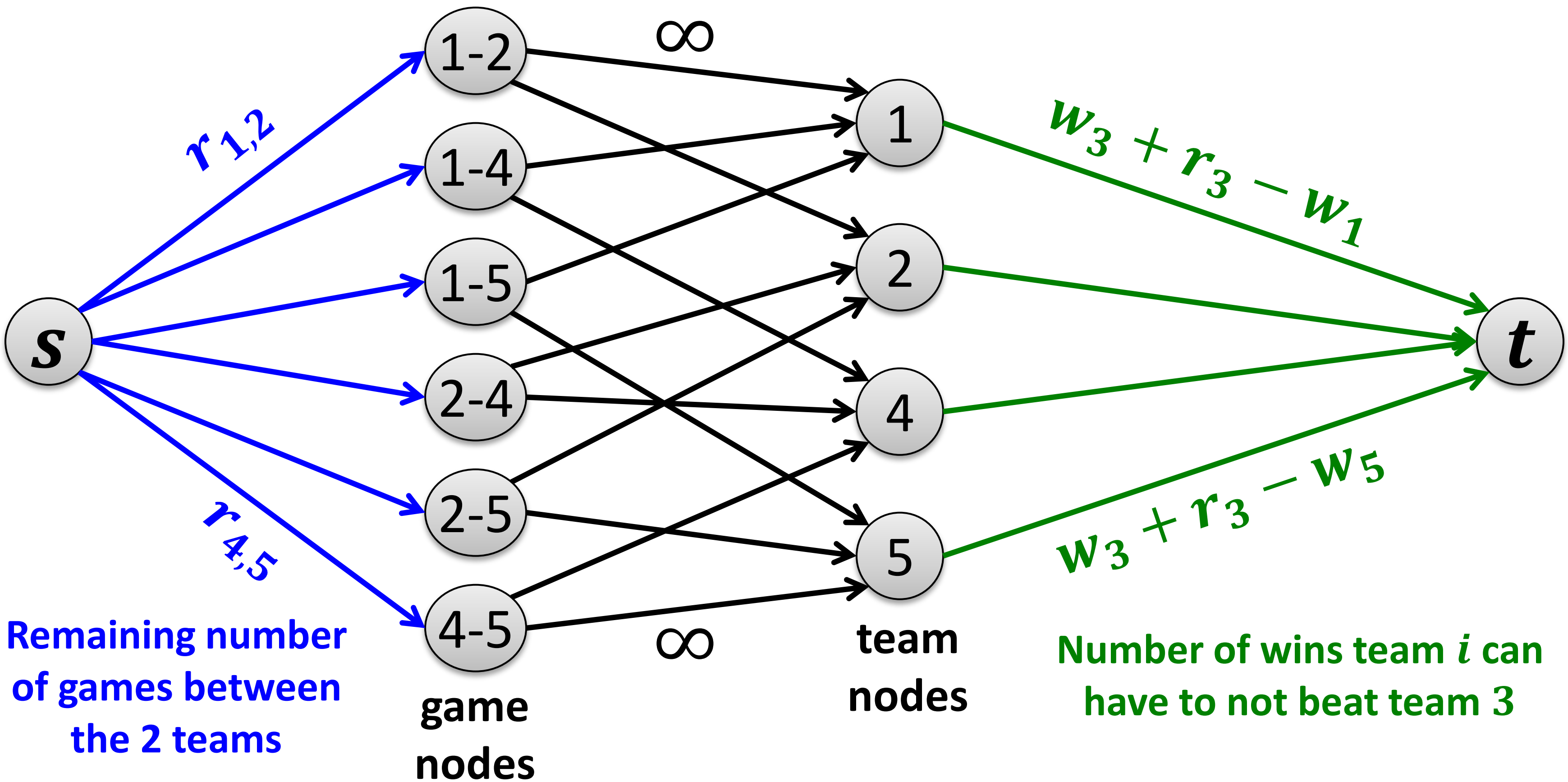
Baseball Elimination

Team i	Wins w_i	Losses ℓ_i	To Play r_i	Against = r_{ij}				
				NY	Balt.	T. Bay	Tor.	Bost.
New York	81	70	11	-	2	4	2	3
Baltimore	79	77	6	2	-	2	1	1
Tampa Bay	79	75	8	4	2	-	1	1
Toronto	76	80	6	2	1	1	-	2
Boston	71	84	7	3	1	1	2	-

- Can Toronto still finish first?
- Toronto can get $82 > 81$ wins, but:
  NY and Tampa have to play 4 more times against each other
  → if NY wins two, it gets 83 wins, otherwise, Tampa has 83 wins
- Hence: Toronto cannot finish first
- How about the others? How can we solve this in general?

Max Flow Formulation

- Can team 3 finish with most wins?



- Team 3 can finish first iff all source-game edges are saturated

Reason for Elimination

AL East: Aug 30, 1996

Team i	Wins w_i	Losses ℓ_i	To Play r_i	Against = r_{ij}				
				NY	Balt.	Bost.	Tor.	Detr.
New York	75	59	28	-	3	8	7	3
Baltimore	71	63	28	3	-	2	7	4
Boston	69	66	27	8	2	-	0	0
Toronto	63	72	27	7	7	0	-	0
Detroit	49	86	27	3	4	0	0	-

- Detroit could finish with $49 + 27 = 76$ wins
- Consider $R = \{\mathrm{NY}, \mathrm{Bal}, \mathrm{Bos}, \mathrm{Tor}\}$
  - Have together already won $w(R) = 278$ games
  - Must together win at least $r(R) = 27$ more games
- On average, teams in R win $\frac{278+27}{4} = 76.25$ games

Reason for Elimination

Certificate of elimination:

$$R \subseteq X, \qquad w(R) := \underbrace{\sum_{i \in R} w_i}_{\text{\#wins of nodes in } R}, \qquad r(R) := \underbrace{\sum_{i,j \in R} r_{i,j}}_{\text{\#remaining games among nodes in } R}$$

Team $x \in X$ is eliminated by R if

$$\frac{w(R) + r(R)}{|R|} > w_x + r_x.$$

Reason for Elimination

Theorem: Team x is eliminated if and only if there exists a subset $R \subseteq X$ of the teams X such that x is eliminated by R.

Proof Idea:

- Minimum cut gives a certificate…
- If x is eliminated, max flow solution does not saturate all outgoing edges of the source.
- Team nodes of unsaturated source-game edges are saturated
- Source side of min cut contains all teams of saturated team-dest. edges of unsaturated source-game edges
- Set of team nodes in source-side of min cut give a certificate R