Chapter 5
Graph Algorithms

Algorithm Theory
WS 2014/15

Fabian Kuhn

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. $f' := \text{augment}(f, P)$;
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 can be implemented to run in $O(mC)$ time.

Proof:

C : value of max flow

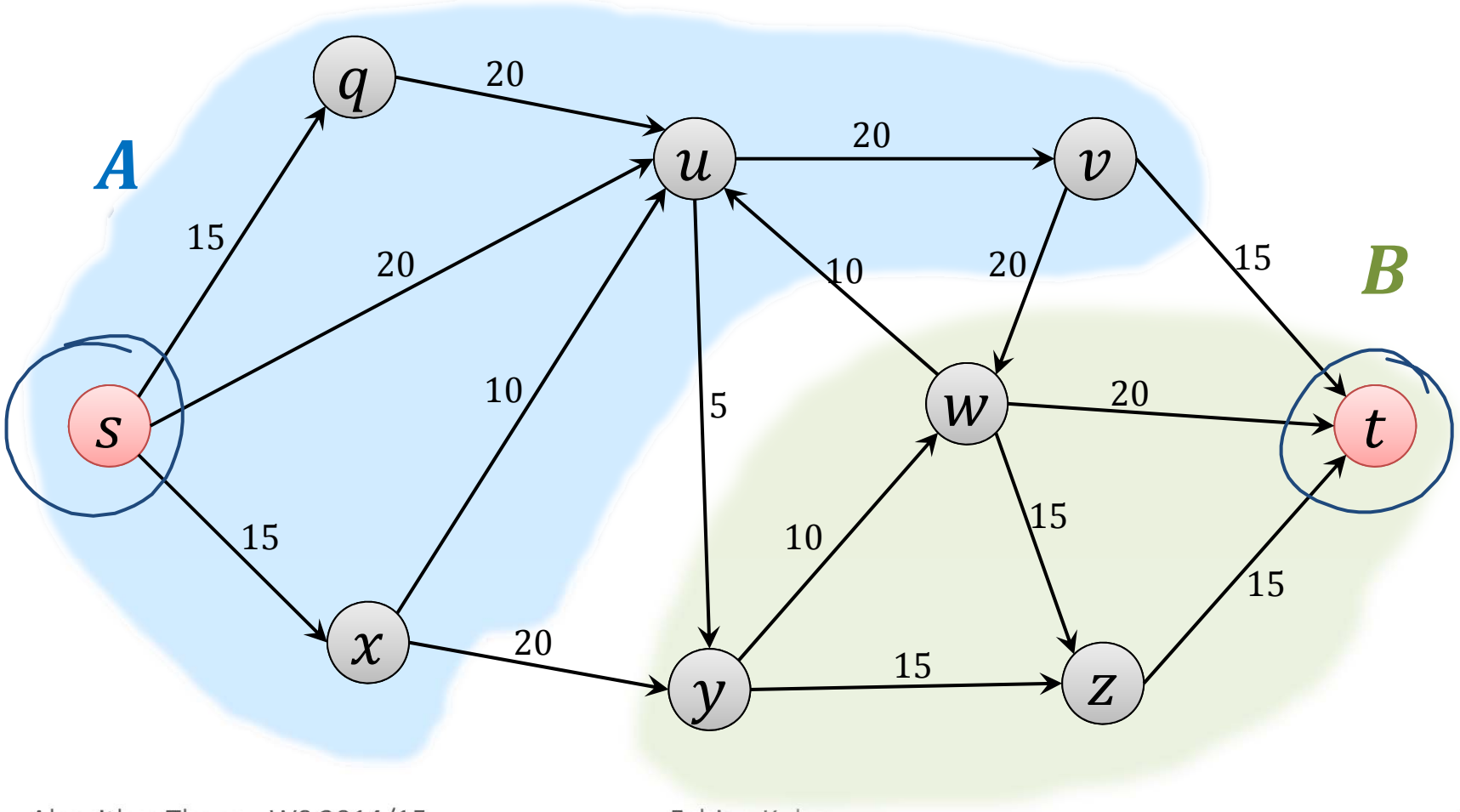
at most C iterations

one iteration: $O(m)$

s-t Cuts

Definition:

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

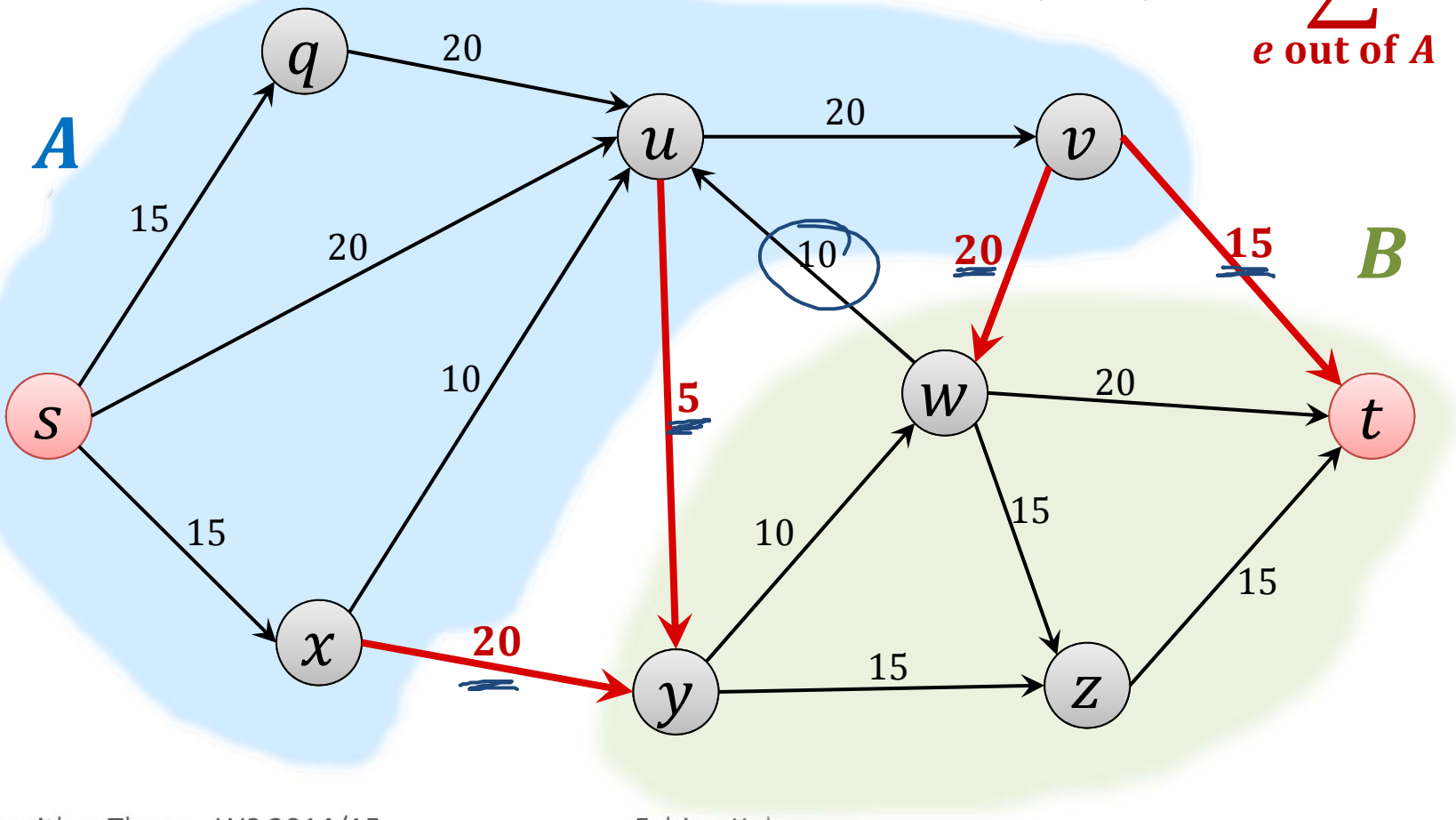


Cut Capacity

Definition:

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

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



Upper Bound on Flow Value

Lemma:

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

Proof:

Ford-Fulkerson Gives Optimal Solution

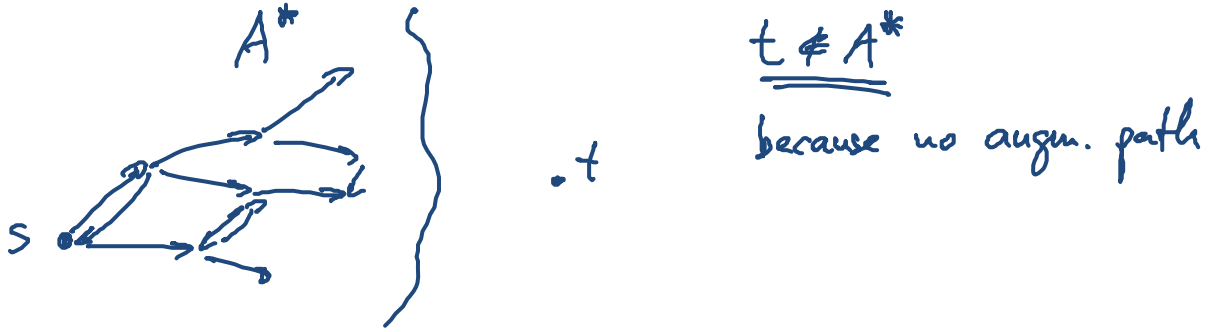
Lemma: If f is an s-t flow such that there is no augmenting path in G_f, then there is an s-t cut (A^*, B^*) in G for which

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

$|f| \leq c(A, B)$ for every s-t-cut (A, B)

Proof:

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



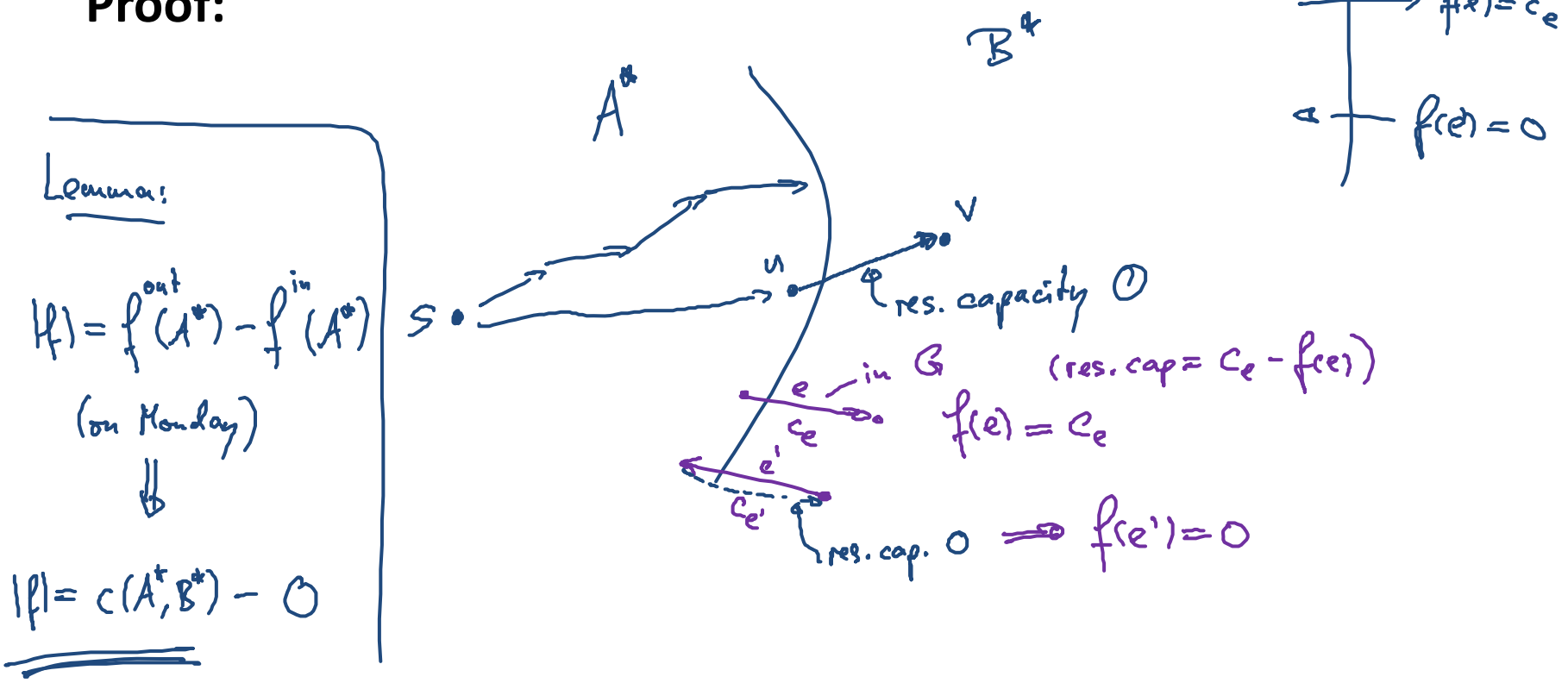
- For $B^* = V \setminus A^*$, (A^*, B^*) is an s-t cut
  - By definition $s \in A^*$ and $t \notin A^*$

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-alg.

↳ cut (A^*, B^*)

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

Lemma before : $|f^*| = c(A^*, B^*)$

$\Rightarrow$ f^* maximum

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 is maximum $\Longrightarrow$ no augm. path

can find cut (A^*, B^*) with $|f| = c(A^*, B^*)$

↳ as before : A^* : set of nodes reachable from s in G_f (over edges of res. cap. > 0)

A^* can be computed in $O(m)$ time using DFS traversal of G_f

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

follows because for every s-t-cut (A,B) $\quad c(A,B) \geq |f|$

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

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 flow f^* and 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:

Ford Fulkerson always maintains an integer flow

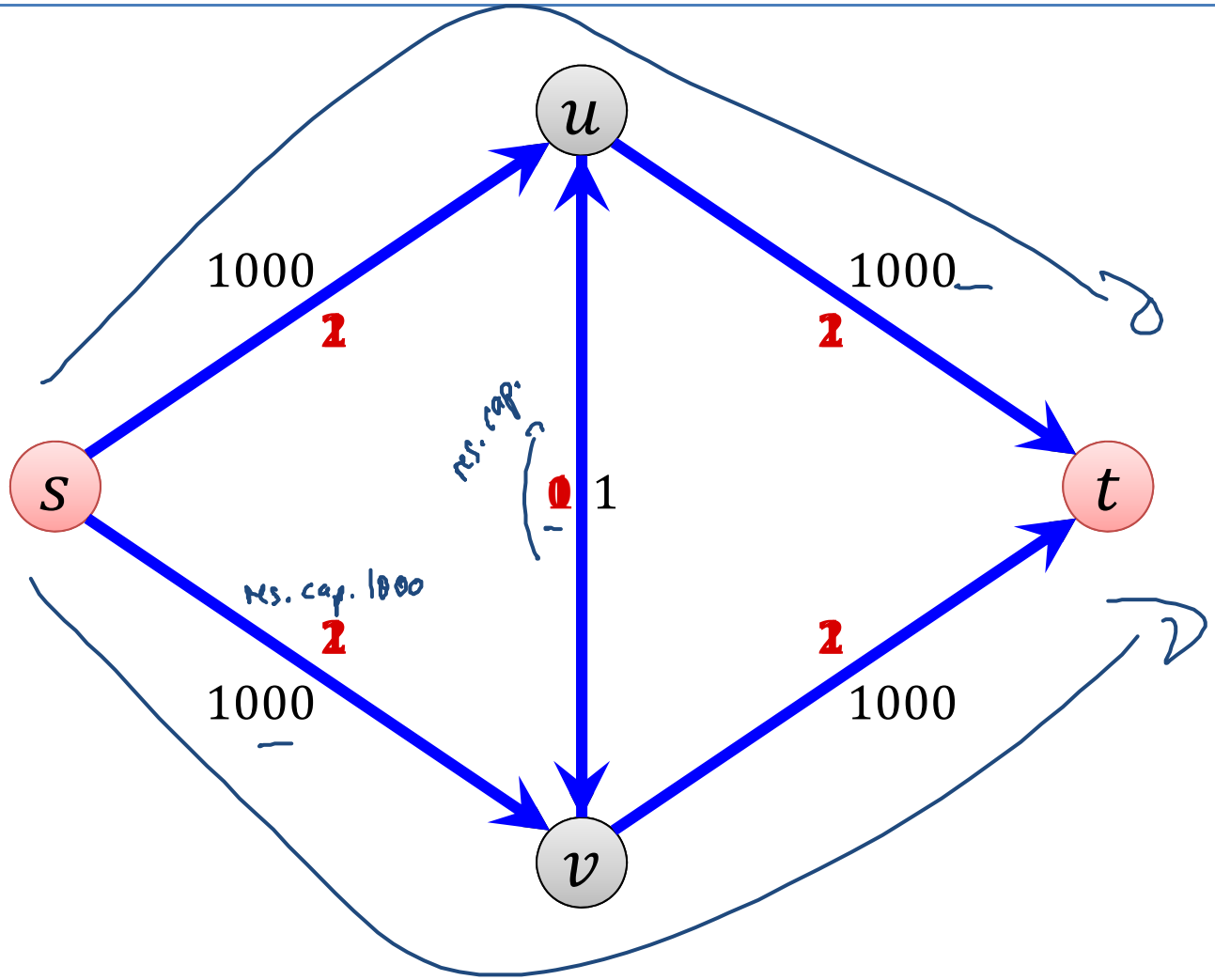
Non-Integer Capacities

What if capacities are not integers?

- rational capacities:
  - 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

$O(m \cdot C)$

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

- **Scaling parameter Δ:**
  (initially, $\Delta = \text{"max } c_e \text{ rounded down to next power of 2"}$)

will be a 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: $\Delta := {}^{\Delta}/_{2}$

How to find such an augm. path:

DFS traversal on G_f by using edges with res. cap. $\geq \Delta$

$O(m)$ time

Scaling Parameter Analysis

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

max cap.

for all e: $c_e \leq C$

$2^{\lfloor \log_2 c_e \rfloor}$ = largest $\Delta \leq \max_e c_e \leq C$

smallest $\Delta \geq 1 = 2^0$

at most $1 + \lfloor \log_2 C \rfloor$ diff. Δ-scaling phases

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

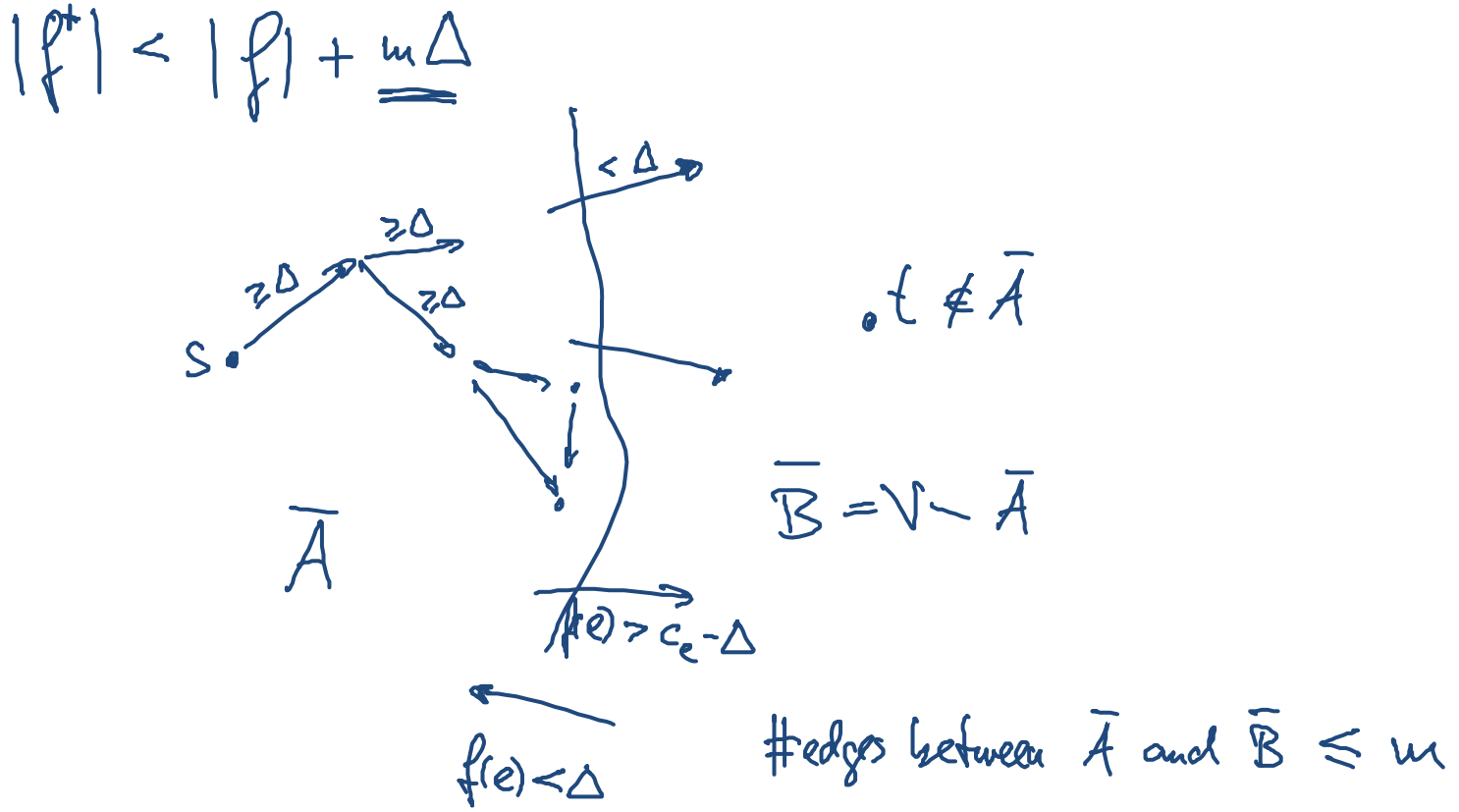
running time:

phases $\cdot$ #iter. per phase $\cdot$ $O(m)$

$O(\log C)$? 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 less than $|f| + m\Delta$.



Length of a Scaling Phase

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

at the beginning of Δ-scaling phase
(at the end of the 2Δ-scaling phase)

$\Rightarrow |f^*| \leq |f| + 2m\Delta$ (lemma before)

each iteration in Δ-scaling phase improves flow 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

$$\rightarrow O(m^2 n)$$

also works for real cap.

#edges, #nodes

Edmonds-Karp Algorithm

Other Algorithms

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

- **Preflow-push algorithm:**
  - Maintains a preflow ($\forall$ nodes: inflow $\geq$ 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: $O(m \cdot n)$**
  - For graphs with $m \geq n^{1+\epsilon}$ (for every constant $\epsilon > 0$) [King,Rao,Tarjan 1992/1994]
  - For sparse graphs with $m \leq n^{16/15-\delta}$ [Orlin, 2013]

for undirected graphs, $(1+\epsilon)$-approximation: $O(m \cdot n^{o(1)})$ [2013]

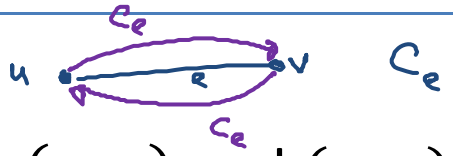
Maximum Flow Applications

- Maximum flow has many applications

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

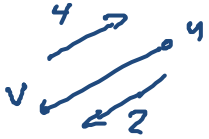
- Examples:
  - related network flow problems
  - computation of small cuts
  - computation of matchings
  - computing disjoint paths
  - scheduling problems
  - assignment problems with some side constraints
  - …

Undirected Edges and Vertex Capacities



Undirected Edges:

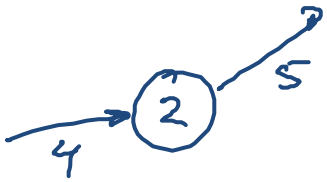
- Undirected edge $\{u, v\}$: add edges (u, v) and (v, u) to network



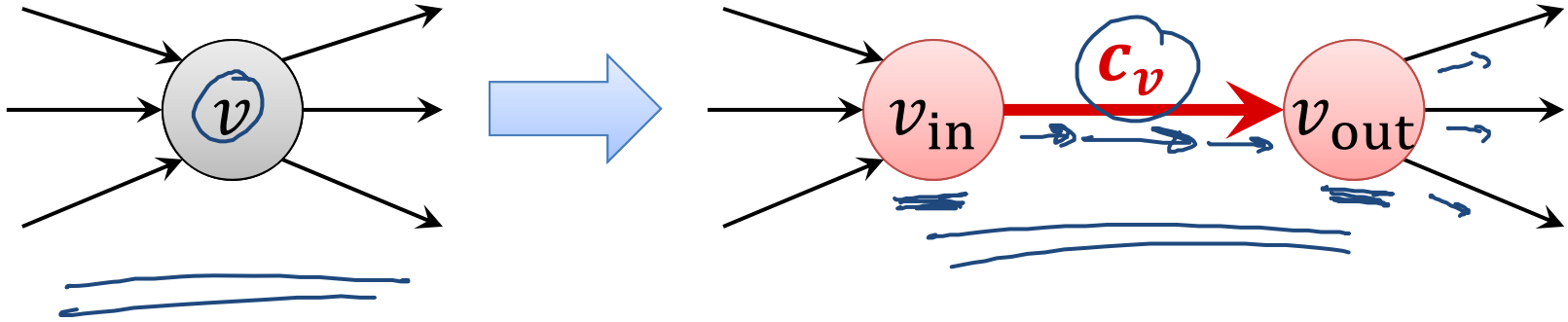
Vertex Capacities:

- Not only edge, 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 $e_v = \{v_{\text{in}}, v_{\text{out}}\}$:

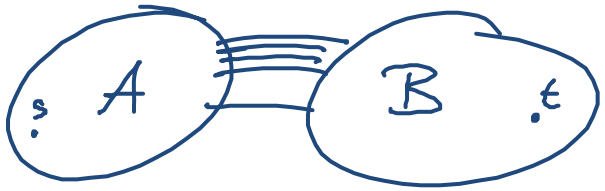


Minimum s-t Cut

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

create a flow network by adding directed edges of cap. 1

size of cut = cap. of cut

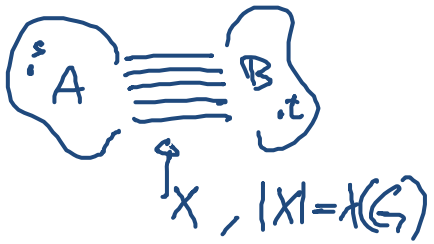
Edge Connectivity

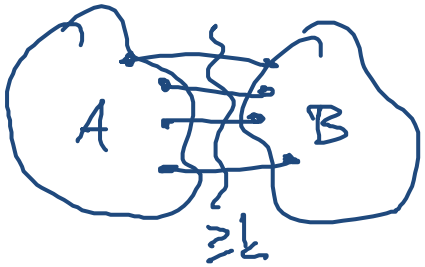
cut partition (A, B), $A, B \neq \emptyset$

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

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

need to remove $\geq k$ edges to disconnect G





edge connectivity $\lambda(G)$
max k s.t. G is k-edge-con.

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 $V \setminus X$

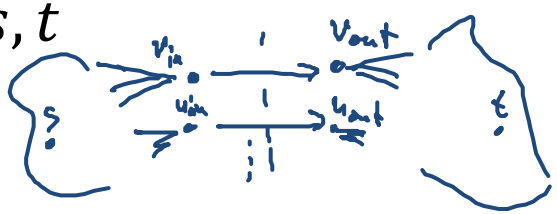
Size of vertex cut: $|X|$

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}:



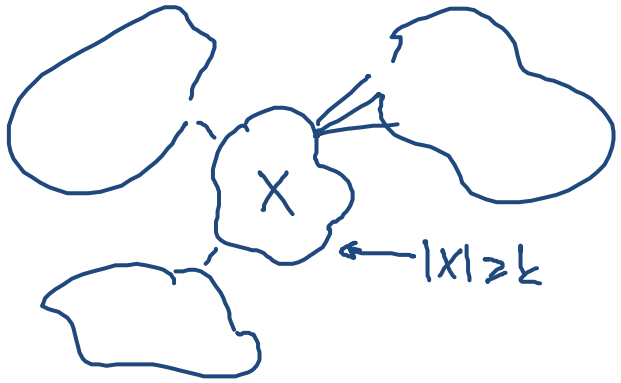
- Min edge cut corresponds to min vertex cut in G

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

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

need to remove $\geq k$ nodes/vertices to disconnect G



$\kappa(G)$: size of smallest set of nodes to disconnect G

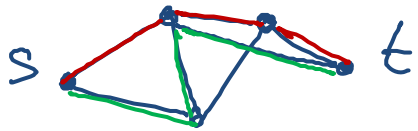
Goal: Compute vertex connectivity $\kappa(G)$ of G
(and node set X of size $\kappa(G)$ that divides G into ≥ 2 parts)

- 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