ORF 522

Linear Programming and Convex Analysis

Network Flows & Ford-Fulkerson

Marco Cuturi

Reminder

- Network Problems
  - a graph topology
  - additional information
- Some canonical problems.
- Light formulation:
  - m nodes, n arcs
  - node-arc incidence matrix $A \in \{0, 1, -1\}^{m \times n}$. last line removed for *l.i.*
- **Tree solutions** for a directed network with n arcs $\mathcal{A}$ and m nodes $\mathcal{N}$:
  - choose $m - 1$ arcs in $\mathcal{A}$ that form a spanning tree $\mathbf{T}$.
  - set flows on arcs not in the tree to **zero**.
  - **flow conditions** determine **uniquely** flow values at the arcs of $\mathbf{T}$.
  - equivalent to basic solutions in standard LP's.

Today

- **Update** and **improve** a tree solution: network simplex.
  - graph interpretation and efficiency
  - implementation speed-ups compared to the simplex
  - complexity
- Generalization to capacitated networks flows.
- The max-flow problem and the **Ford-Fulkerson** algorithm.

Network Simplex

Recapitulating

- Basic feasible solutions $\Leftrightarrow$ **Tree feasible** solutions
  - **intuition**: a set of edges that form trees lead to invertible matrices.
  - why? because they are **triangular** with suitable reordering.
  - if not a tree, there is a cycle in a set of edges $\mathbf{I}$. what happens to $B_{\mathbf{I}}$?
- The basic solution $\mathbf{f_T}$ can be computed directly. Just by starting from the leaves of $\mathbf{T}$ and going up to the root.
- No need to invert $B_{\mathbf{T}}$.
- **Degeneracy**: some flows in arcs belonging to $\mathcal{T}$ might be 0. The same flow might correspond to different trees.

Changing the basis, changing the tree

- Remember the primal simplex:
  - Given $\mathbf{I}$, identify an **entering** **column or variable** w.r.t. reduced costs.
  - identify an **exit** **column variable** that conserves feasibility.
- **Same idea** here:
  - Given $\mathbf{T}$, identify an **entering** **arc** of $\mathcal{A} \setminus \mathbf{T}$ that will improve the objective
  - Find an **exit** **arc** of $\mathbf{T}$ to remove that ensures we still have **a feasible tree**

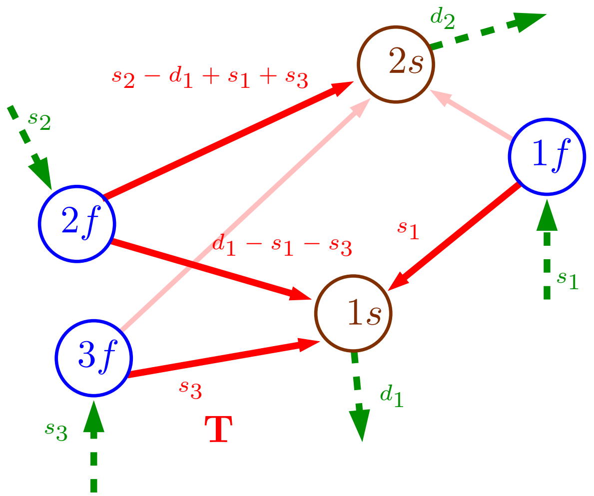
Changing the basis, changing the tree

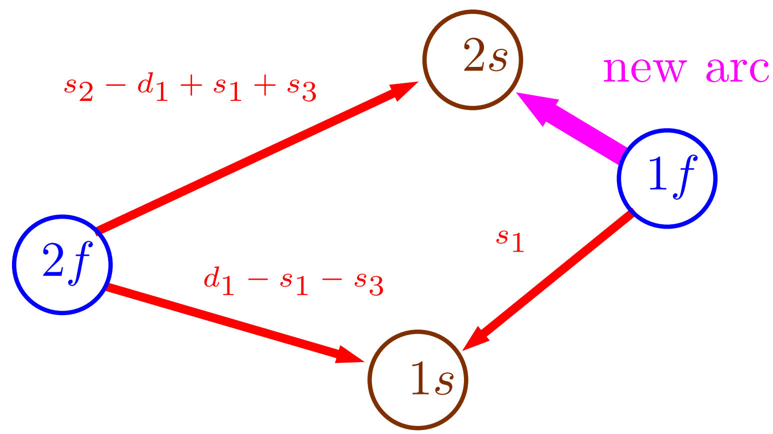
- More precisely:
- Pick an arc (i,j) not in $\mathbf{T}$. Add it to the tree.
- Obtain a cycle C that includes (i,j).
- Choose the orientation of C such that $i, (i,j), j$ is in C.
- (i,j) is a **forward** arc of C. Label other arcs as **F**orward or **B**ackward.
- Push θ of the circulation C into f. The flow vector $\mathbf{f}$ becomes
  - $f_a \leftarrow \begin{cases} f_a + \theta & \text{if } a \in F, \\ f_a - \theta & \text{if } a \in B, \\ f_a & \text{otherwise.} \end{cases}$
  - To ensure **feasibility**, that is nonnegativity, the largest possible value for θ is

$$\theta^\star = \min_{(k,l)\in B} f_{k,l} \text{ or } \infty \text{ if } B = \emptyset. \tag{1}$$

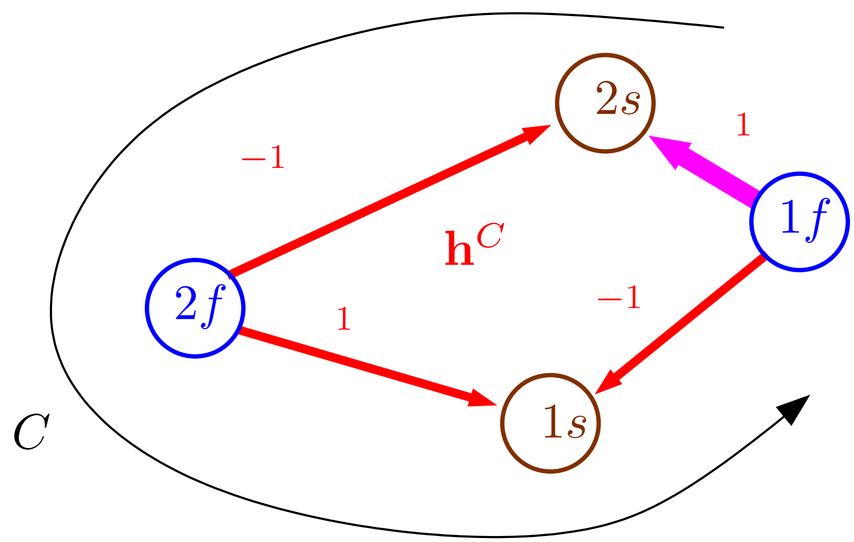
  - If (k,l) is the argmin, remove it from $\mathbf{T}$ and we get a **new tree flow**.

Changing the basis, changing the tree



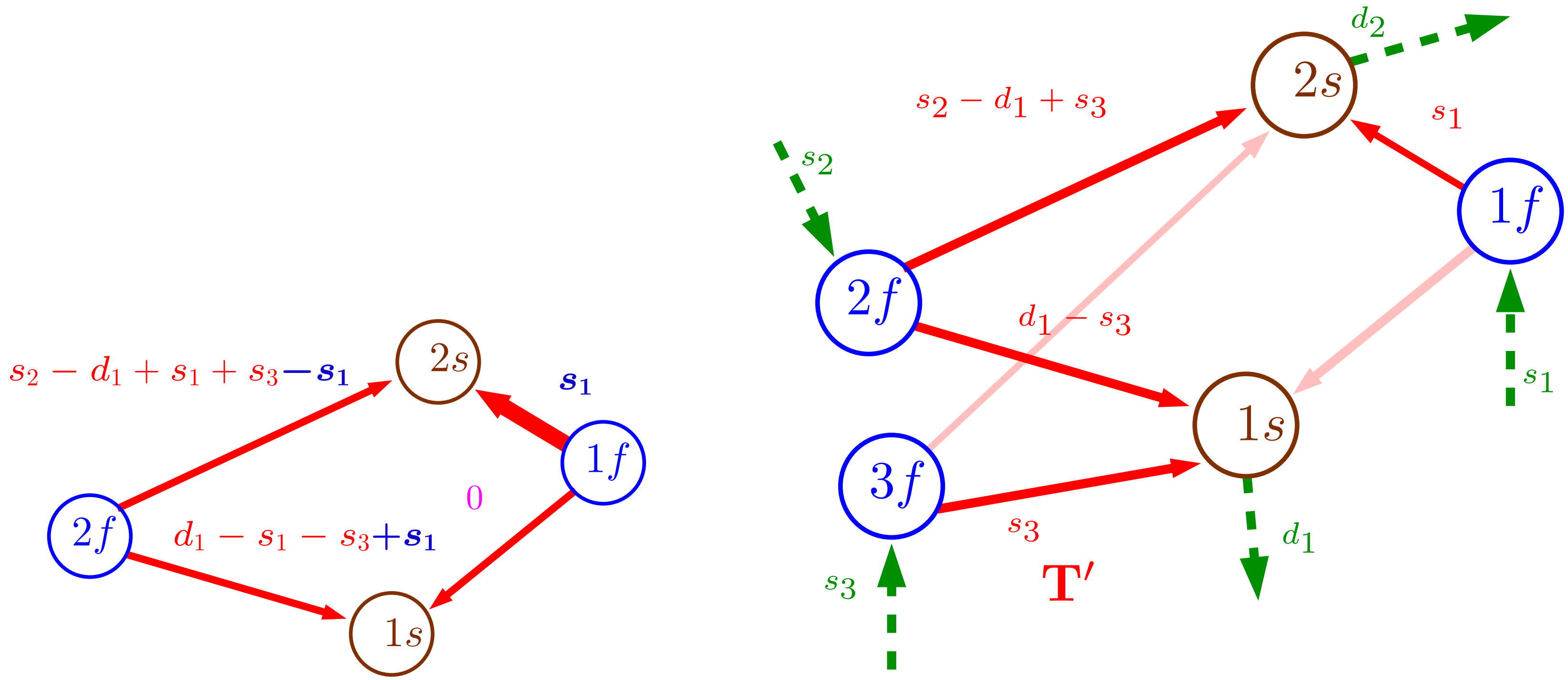


- The amount $\theta^\star$ and (k,l) depends **only** on the actual values of flows of **backward** arcs: s_1 and $s_2 - d_1 + s_1 + s_3$.



Changing the basis, changing the tree

- suppose s_1 is smaller. Then $\theta^* = s_1$ the new values at the cycle are:



- We have a new **tree feasible solution**

- If we push θ units of flow, the objective changes by

$$\theta^* \underbrace{\left(\sum_{(k,l)\in F} c_{k,l} - \sum_{(k,l)\in B} c_{kl} \right)}_{\text{reduced cost}}.$$

Reduced Cost Coefficient For an Arc

- This coefficient provides a criterion to select entering **arc** (i, j). We used a cycle C to define it, is it **unique**?
- For each arc (i, j) of $\mathcal{A} \setminus \mathbf{T}$, there is only one cycle (up to shifts) obtained by adding arc (i, j) and which has (i, j) as a forward arc. (why?)
- We can thus define a vector $\mathbf{r}$ of size n, $r_a = 0$ for $a \in \mathbf{T}$ and for $(i, j) \notin \mathbf{T}$,

$$r_{(i,j)} = \left(\sum_{(k,l)\in F} c_{k,l} - \sum_{(k,l)\in B} c_{k,l} \right).$$

  which is called the **reduced cost coefficient** vector.

- Same quantity than if we had gone through the simplex computations.
- Yet looks more tedious to compute in this form... $\rightarrow$ use duality

Reduced Cost Computation

- Recall the reduced cost vector formula:

$$\mathbf{r} = \mathbf{c} - A^T \mu,$$

- where the **dual vector** μ corresponds to the base $\mathbf{I}$, namely $B_{\mathbf{I}}^{-1}\mathbf{c}_{\mathbf{I}}$.
- $\mu \in \mathbf{R}^{m-1}$ (# nodes -1).
- $A^T \in \mathbf{R}^{n\times(m-1)}$ has n rows with only a 1, a -1 and 0's (except for the last one).
- We thus have

$$r_{(i,j)} = \begin{cases} c_{(i,j)} - (\mu_i - \mu_j), & \text{for } i \neq j \leq m-1, \\ c_{(i,j)} - \mu_i, \text{ for } j = m \\ c_{(i,j)} + \mu_j, \text{ for } i = m. \end{cases}$$

Reduced Cost Computation

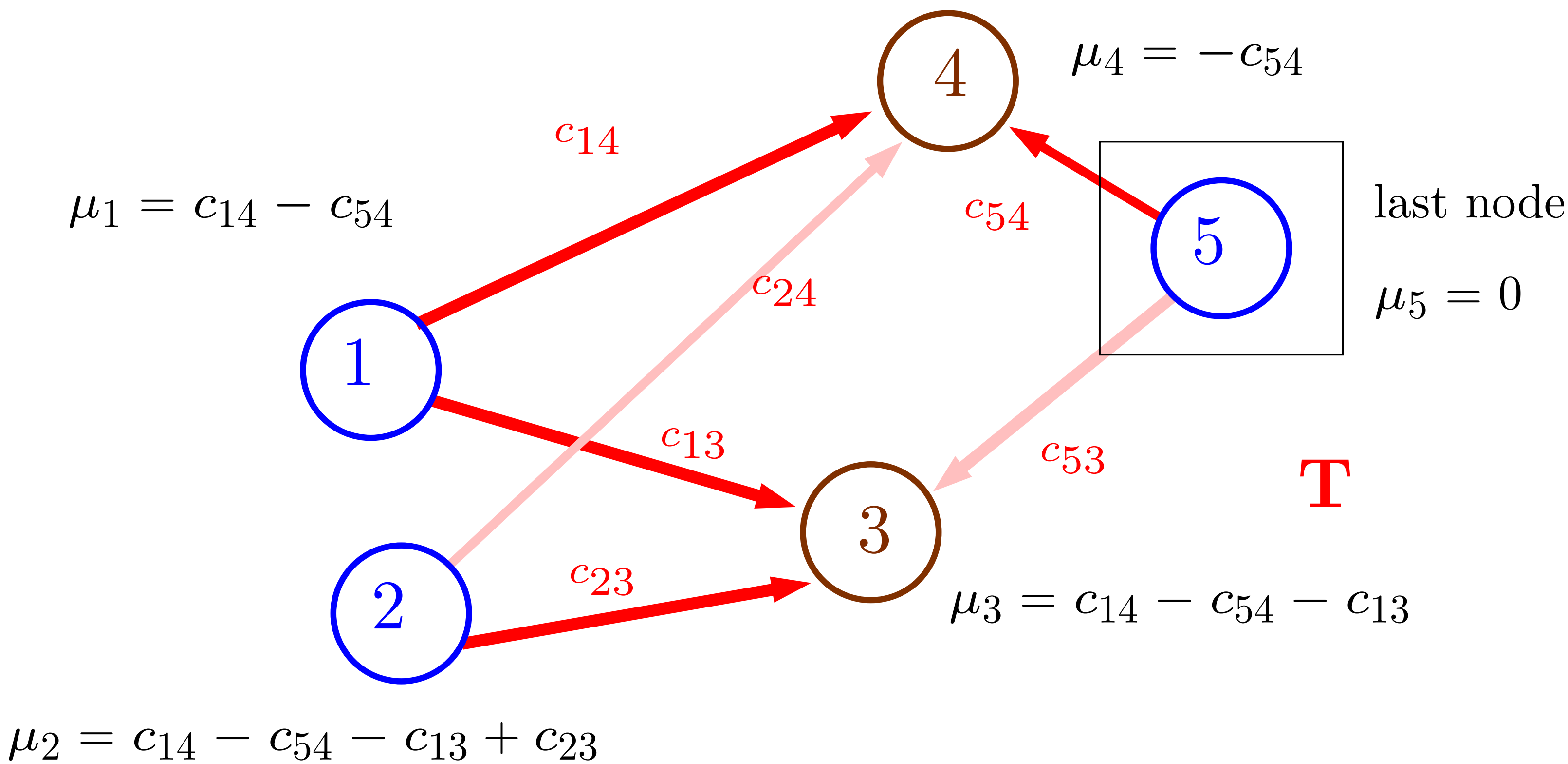
- We define the mth coordinate of μ, $\mu_m = 0$.
- We then have
$$\forall (i,j) \in \mathcal{A}, \quad r_{(i,j)} = c_{(i,j)} - (\mu_i - \mu_j). \tag{2}$$
- How do we compute $\mu = B_{\mathbf{I}}^{-1}\mathbf{c}_{\mathbf{I}}$?
- We use the fact that the reduced cost coefficient of a basic variable is zero, $i.e.$

$$\forall (i,j) \in \mathbf{T}, \quad \mu_i - \mu_j = c_{ij}$$

we have $m-1$ linear relationships for $m-1$ unknown variables..

Reduced Cost Computation

- In practice, start from the last node and cascade through all edges in $\mathbf{T}$.



- Once this is done, compute $\mathbf{r}$ using Equation (2), only for arcs $(i, j) \notin \mathbf{T}$

Recapitulation

- **Input:** directed graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$, cost vector $\mathbf{c}$.
- **Algorithm**: minimize $\mathbf{c}^T\mathbf{f}$ under flow constraints, including nonnegativity.
  - Start with a feasible tree $\mathbf{T}$.
  - Set $f_a, a \in \mathbf{T}$ following the flow conservation equations. For $a \notin \mathbf{T}, f_a = 0$.
  - Compute dual variables $\mu_1, \cdots, \mu_{m-1}$ by starting from the root $\mu_m = 0$.
  - Compute reduced costs: $r_{ij} = c_{ij} - (\mu_i - \mu_j)$ for $(i, j) \notin \mathbf{T}$.
  - If $r_a \geq 0$ for all arcs of $\mathcal{A} \setminus \mathbf{T}$, $\mathbf{T}$ is **optimal**.
  - otherwise, choose e in $\{a \in \mathcal{A} \setminus \mathcal{T} | \, r_a < 0\}$ and add it to $\mathbf{T}$.
  - Set the cycle C such that a is a **forward arc** of C.
  - Determine θ^* according to Equation (1).
  - Update the flow vector using $\mathbf{h}^C$, namely

$$f_a \leftarrow \begin{cases} f_a + \theta^* & \text{if } e \in F. \\ f_a - \theta^* & \text{if } e \in B. \\ f_a, & \text{otherwise.} \end{cases}$$

Computational Insights

Unimodular Matrices: Another Property

Definition 1. *A square* ***integer*** *matrix is* **unimodular** *if its determinant is* -1 *or* $+1$

- Easy to remark that for a choice of edges $\mathbf{T}$ that corresponds to a tree $B_{\mathbf{T}}$ is unimodular.

Definition 2. *The* ***inverse of a unimodular*** *matrix is* **unimodular***.*

- **Proof** ?
  - For a matrix A, Minor $M_{ij} = \det([A_{kl}]_{k\neq,l\neq j})$, Cofactor $C_{ij} = (-1)^{i+j} M_{ij}$.
  - Cramer's rule: $A^{-1} = \frac{1}{\det(A)} C^T$.
- Hence if $\mathbf{b}$ is integer valued, all tree flows are integers!
- If $\mathbf{b}$ is rational, multiply by GCD.
- In all cases, substantial gain in memory for practical implementations.

Initialization of the network simplex

- Find a spanning tree? off-the-shelf algorithms: depth-first/breadth-first searches, worst-case complexity of $O(n+m)$.
- Initialization: find a **feasible** spanning trees. **Phase I** type method:
  - Start with origins, choose forward arcs, and destinations, with backward arcs.
  - Build F-paths from origin and B-paths from destinations until they meet.
  - Complete to form a **spanning tree** that connects all nodes.
  - Assign values of with **flow conservation equations**. Set $\mathcal{A}' = \mathcal{A}$.
  - If $f_{(i,j)}$ for an arc is negative, add if necessary $\mathcal{A}' \leftarrow \mathcal{A}' \cup (j,i)$,
  - set $f_{(j,i)} \leftarrow -f_{(i,j)}$ and $f_{(i,j)} \leftarrow 0$.
  - Drive out artificial arcs: min. $\omega = \sum_{a\in\mathcal{A}'} \delta_{a\notin\mathcal{A}} f_a$, use the **network simplex**.
  - If $\omega > 0$ then **infeasibility**.
  - If $\omega = 0$, $f_a = 0$ for a in $\mathcal{A}' \setminus \mathcal{A}$ and we have an **initial feasible tree**.
- M-type methods are also possible:
  - add artificial edges with very high costs that link pairs of source-destinations
  - complete the tree, incorporate these costs in the overall cost criterion.

Complexity of the network simplex

- Given a tree $\mathbf{T}$, the time consuming steps at **each iteration**:
  - Computing dual variables takes $O(m)$ operations,
  - Computing reduced costs takes $O(n)$ operations,
  - Updating flows in $\mathbf{T}$ takes $O(m)$ operations.
- since $n \geq m - 1$, $O(n)$ operations in total.
- Compares favorably with the $O(mn)$ operations of the simplex pivot.

- What about the **total number** of iterations?

Complexity of the network simplex

- Open questions: how many solutions at most?
  - For LP's, only approximations: $\#\{$extreme points of the feasible set$\}$.
  - Cayley: complete undirected graph of n nodes $\Rightarrow n^{n-2}$ spanning trees.

- For the more general case, **Kirchhoff formula**:
  - **Laplacian matrix** L of undirected graph $(\mathcal{N}, \mathcal{E})$:
    - ▷ L is a $m \times m$ matrix (nodes $\times$ nodes).
    - ▷ $l_{i,i} = \text{deg}(i), \quad l_{i,j} = \begin{cases} -1 \text{ if } \{i,j\} \in \mathcal{E} \\ 0 \text{ otherwise} \end{cases}$
    - ▷ L is not invertible. $\lambda_1 = 0$ is an eigenvalue. The multiplicity of 0 gives the number of **connected subgraphs** of $\mathcal{G}$.
  - Kirchhoff: the number $t(\mathcal{G})$ of **spanning trees** of $\mathcal{G}$ is **equal to**

$$t(\mathcal{G}) = \frac{1}{m}\lambda_2\lambda_3\cdots\lambda_m.$$

- **Bottom line**: *Usually* complexity of $O(m)$ but there exist examples where the network simplex takes exponential number of steps.

Capacitated Problems

Network Simplex for Capacitated Problems

- We now deal with the general capacitated case, *i.e.*

$$d_a \leq f_a \leq u_a, a \in \mathcal{A}$$

- By basic solution we usually mean:
  - Select a tree $\mathbf{T} \subset \mathcal{A}$.
  - Set the flow values to zero for arcs in $\mathcal{A} \setminus \mathbf{T}$.
  - Fill in values for $\mathbf{f_T}$ through flow conservation.
- In the **capacitated** case, this will become
  - Select a tree $\mathbf{T} \subset \mathcal{A}$.
  - For arcs in $\mathcal{A} \setminus \mathbf{T}$, split them into two subsets $\mathbf{U}$ and $\mathbf{D}$.
    - ▷ arcs in $\mathbf{U}$ have maximal flows $f_a = u_a$.
    - ▷ arcs in $\mathbf{D}$ have minimal flows $f_a = d_a$.
  - Fill in values for $\mathbf{f_T}$ through flow conservation equations.

Network Simplex for Capacitated Problems

- Suppose a tree $\mathbf{T}$ is given, with other arcs in $\mathbf{U}$ or $\mathbf{D}$.
- How should we look for the arcs to add e / remove r from the basis $\mathbf{T}$?
- As before, compute **reduced costs vector** for arcs of $\mathbf{U}$ **and** $\mathbf{D}$.
- If any arc a in $\mathbf{D}$ has a **negative** reduced cost,
  - choose cycle C that contains a as a **forward** arc.
  - pushing θ units of flow through that cycle we improve the objective.
- If any arc a in $\mathbf{U}$ has a **positive** reduced cost,
  - choose cycle C that contains a as a **backward** arc.
  - pushing θ units of flow through that cycle we improve the objective.

Network Simplex for Capacitated Problems

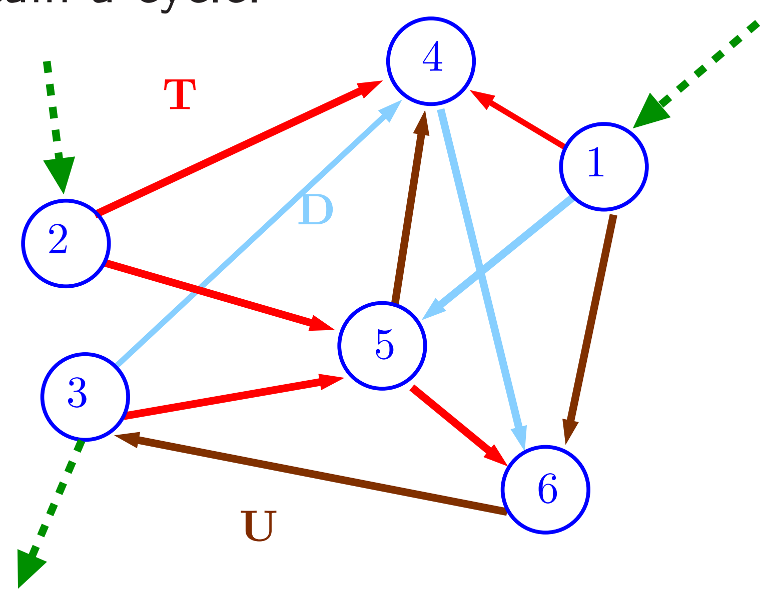
- In both cases, objective improve. We need to be sure feasibility is ensured.
- Whatever the considered cycle,
  - arcs in F see their flow **in**creased: check $\leq u.$.
  - arcs in B see their flow **de**creased: check $\geq d.$.
- hence

$$\theta^* = \min\left\{\min_{a\in B}(f_a - d_a), \min_{a\in F}(u_a - f_a)\right\}. \tag{1}$$

- There will be (at least) one arc r of $\mathbf{T}$ which will be saturated, either equal to d_r or u_r.
- r will leave $\mathbf{T}$ and enter $\mathbf{U}$ or $\mathbf{D}$.
- This arc will be *usually* replaced by a which was selected because of its reduced cost coefficient.
- Why *usually*? because in some cases a flow that was equal to u_i we want to enter $\mathbf{T}$ might become equal to d_i. We've added/removed the same flow in one operation.

Capacitated Network Simplex

- **Input:** directed graph $\mathcal{G}(\mathcal{N}, \mathcal{A})$, cost vector $\mathbf{c}$, capacities $\mathbf{d}, \mathbf{u}$.
- **Algorithm**: minimize $\mathbf{c}^T\mathbf{f}$ under flow and capacities constraints.
  - Start with a tree $\mathbf{T}$ with BFS, and a partition $\mathbf{D}, \mathbf{U}$ of $\mathcal{A} \setminus \mathbf{T}$.
  - $f_a = d_a$ for arcs in $\mathbf{D}$, $f_a = u_a$ for arcs in $\mathbf{U}$, and f_a feasible following the flow conservation equations.
  - Compute dual variables $\mu_1, \cdots, \mu_{m-1}$ by starting from the root $\mu_m = 0$.
  - Compute reduced costs: $r_{ij} = c_{ij} - (\mu_i - \mu_j)$ for $(i, j) \notin \mathbf{T}$.
  - If $r_a \geq 0$ for all arcs in $\mathbf{D}$ and $r_a \leq 0$ for all arcs in $\mathbf{U}$, $\mathbf{T}$ is **optimal**.
  - otherwise, choose e in either $\{a \in \mathbf{D} |\, r_a < 0\}$ or $\{a \in \mathbf{U} |\, r_a > 0\}$. By adding e to $\mathbf{T}$ we obtain a cycle.



Capacitated Network Simplex

- Choose the cycle C such that
  - ▷ e is a **forward arc** of C if e was in $\mathbf{D}$,
  - ▷ e is a **backward arc** of C if e was in $\mathbf{U}$.
- Determine θ^* according to Equation (1).
- Update the flow vector using $\mathbf{h}^C$, namely

$$f_a \leftarrow \begin{cases} f_a + \theta^* & \text{if } a \in F. \\ f_a - \theta^* & \text{if } a \in B. \\ f_a, & \text{otherwise.} \end{cases}$$

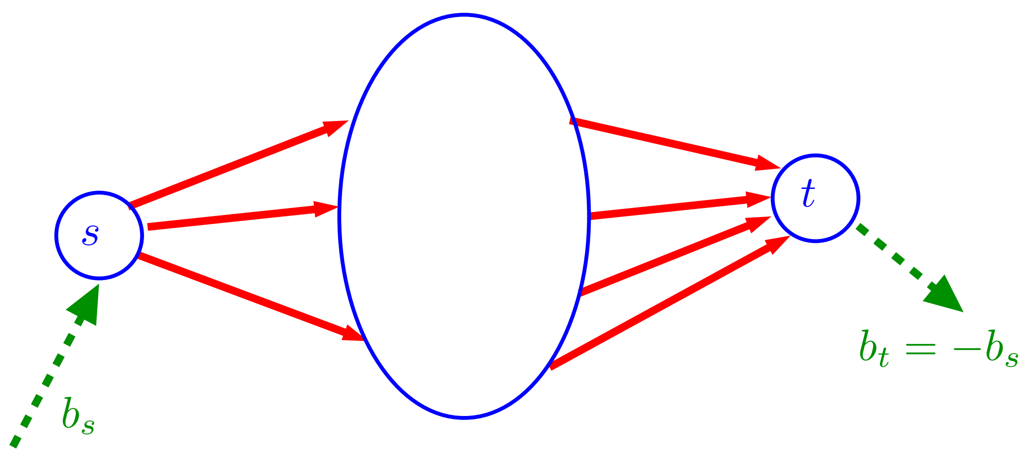
- Update the sets $\mathbf{T}, \mathbf{U}, \mathbf{D}$ and repeat.

Maximum-flow
and
the Ford-Fulkerson Algorithm

Direct formulation

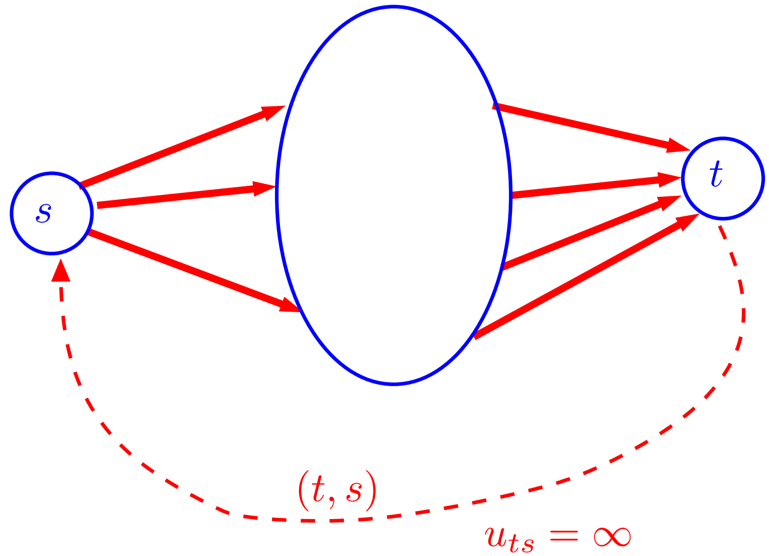
We considered the following flow example:

- m nodes,
- n arcs,
  - Each arcs a carries a flow f_a its flow.
  - Each edge has a bounded capacity (pipe width) $0 \leq f_a \leq u_j$
- One source node s, one sink node t. $b_s > 0, b_t < 0, b_s + b_t = 0$. The other supplies are zero.
- A possible formulation would be to maximize b_s given all flow constraints:



Network Flow Formulation

- Maximizing a supply is not exactly what we considered in our programs.
- We add an artifical edge $a = (t, s)$ instead,



and reformulate the problem as

$$\begin{array}{ll} \text{minimize} & -f_{t,s} \\ \text{subject to} & A\mathbf{f} = 0, \\ & \mathbf{0} \leq \mathbf{f} \leq \mathbf{u}. \end{array}$$

- Using this reformulation, solve solve with the network simplex.

Network Flow Formulation

- **More efficient** algorithms exist. We look for the biggest b_s possible.
- Let's start with the definition of **augmenting paths**

  Definition 3. *Let* $\mathbf{f}$ *be a feasible flow vector to the max-flow problem. An* ***augmenting path*** *is a path from* s *to* t *such that* $\boldsymbol{f_a < u_a}$ *for all forward arcs* F *and* $\boldsymbol{f_a > 0}$ *for all backward arcs* B *of the path.*

- An augmenting path is also called an *unsaturated* path.
- With an augmenting path P, we can change the flow along every arc:
  - increase by θ for forward arcs,
  - decrease by θ for backward arcs.
- The maximal increase/decrease is

$$\theta(P) = \min \left\{ \min_{a \in F}(u_a - f_a), \min_{a \in B} f_a \right\}.$$

Ford-Fulkerson Algorithm

- Here is a high-level description, we check details later

  1. Start with a feasible flow $\mathbf{f}$. The zero-flow is valid at first iteration.
  2. **Search for an augmenting path P.**
  3. If no augmenting path can be found, terminate.
  4. If an augmenting path can be found, then
     (a) if $\theta(P) < \infty$ push $\theta(P)$ units of flow along P.
     (b) if $\theta(P) = \infty$, terminate.

- **Remark**: if all **capacities** are **integer** or infinite, and the algorithm is initialized with an **integer feasible flow**, then if the optimum is finite the algorithm terminates after a finite number of steps.

- **Why**? flow increases by $\theta(P) \in \mathbb{N}, \theta(P) > 1$. If optimum the algorithm must stop in a finite number of steps.

- Can be generalized to rational numbers.

Search for an augmenting path P

- The search itself is known as the **labeling** algorithm.
- The labeling algorithm is a simple brute-force search that explores the graph from s to t looking for such paths.
- Some intuitions:
  - Suppose we have an augmenting path from s to an **intermediary** node i. if,
    - ▷ $(i, j) \in \mathcal{A}$ and $f_{(i,j)} < u_{ij}$ or
    - ▷ $(j, i) \in \mathcal{A}$ and $f_{(j,i)} > 0$,

    then we can start looking from j to find an augmenting path.
- The process of examining all nodes j neighboring node i is called **scanning** i.
- **Idea**:
  - keep track in I of **labelled** nodes, that is nodes for which an augmenting path from s to i exists, which **have not been scanned yet**.
  - **scan** the nodes of I, remove them and move forward along the graph by adding eventually **labelled** nodes.

The Labeling algorithm

- **Initialize** the algorithm with $I = \{s\}$.
- Loop:

(i) If $I = \emptyset$ there is no augmenting path.
(ii) If node $t \in I$ terminate with an augmenting path.
(iii) Otherwise scan any element of I, say i:
  - Remove i from I.
  - Look for all neighbors j of i that satisfy the augmenting path condition, that is
    - ▷ if $(i, j) \in \mathcal{A}$ and $f_{(i,j)} < u_{ij}$ or
    - ▷ if $(j, i) \in \mathcal{A}$ and $f_{(j,i)} > 0$.
    - ▷ Add these nodes j's into I.

- Complexity: $O(\#(\mathcal{A}))$

Cuts

- We introduce cuts, both to prove the convergence of Ford-Fulkerson and introduce a parallel with duality.
- An $(s-t)$ cut is a subset S of nodes such that $s \in S$ and $t \notin S$.
- The capacity of the cut is the sum of the capacities of the arcs that cross from S to its complement $T = \mathcal{N} \setminus S$,

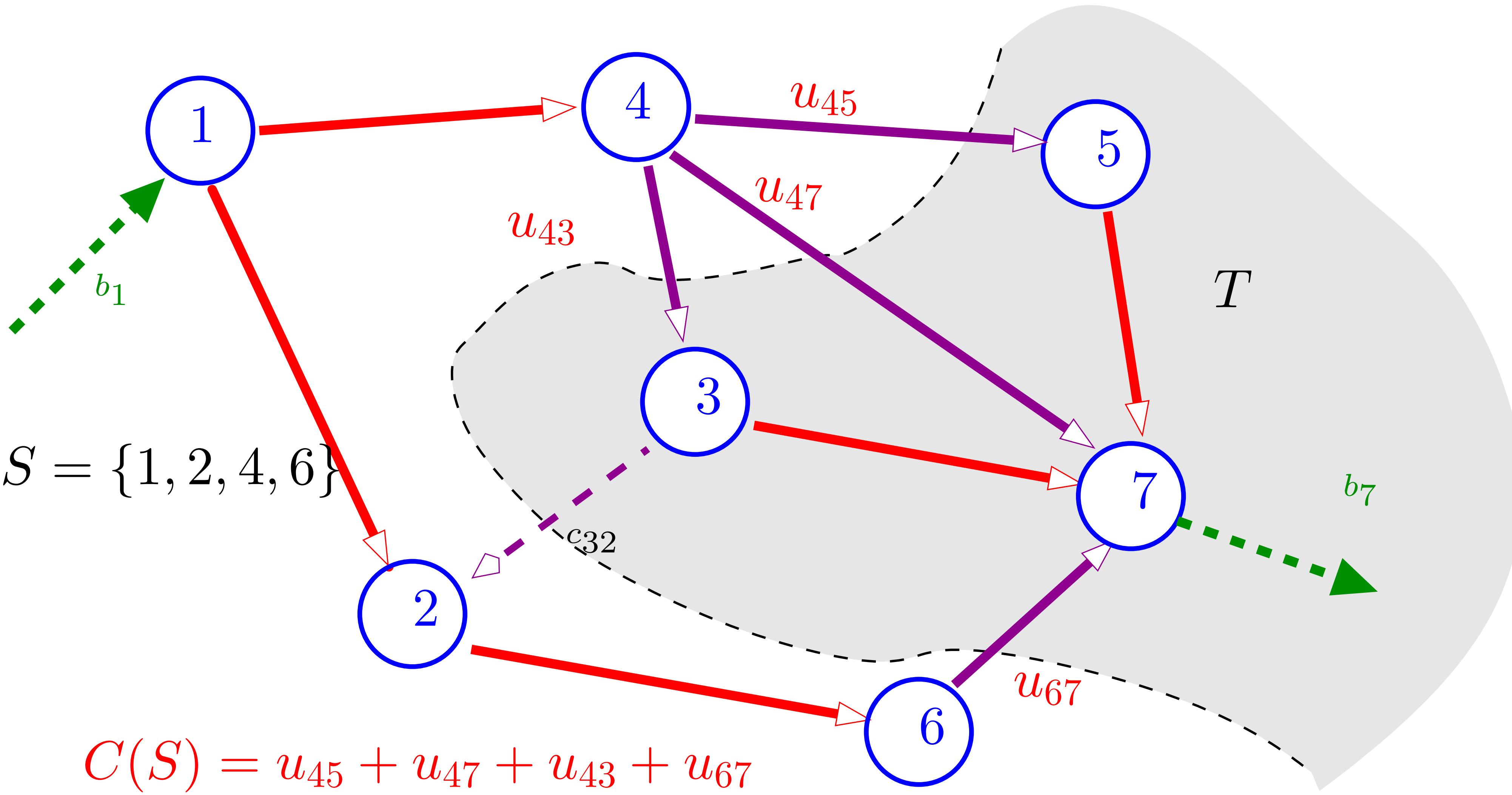
$$C(S) = \sum_{(i,j)\in\mathcal{A} \mid i\in S, j\in T} u_{(i,j)}$$

- Additionally, any overall flow from s to t crosses at different points the line between a node $i \in S$ and a node $j \in T$.
- Hence **for every cut S** the flow supplied to the network b_s is upperbounded by

$$b_s \leq C(S),$$

- cuts provide a family of upperbounds. What about the minimal cut?... see slides on duality.

Cuts

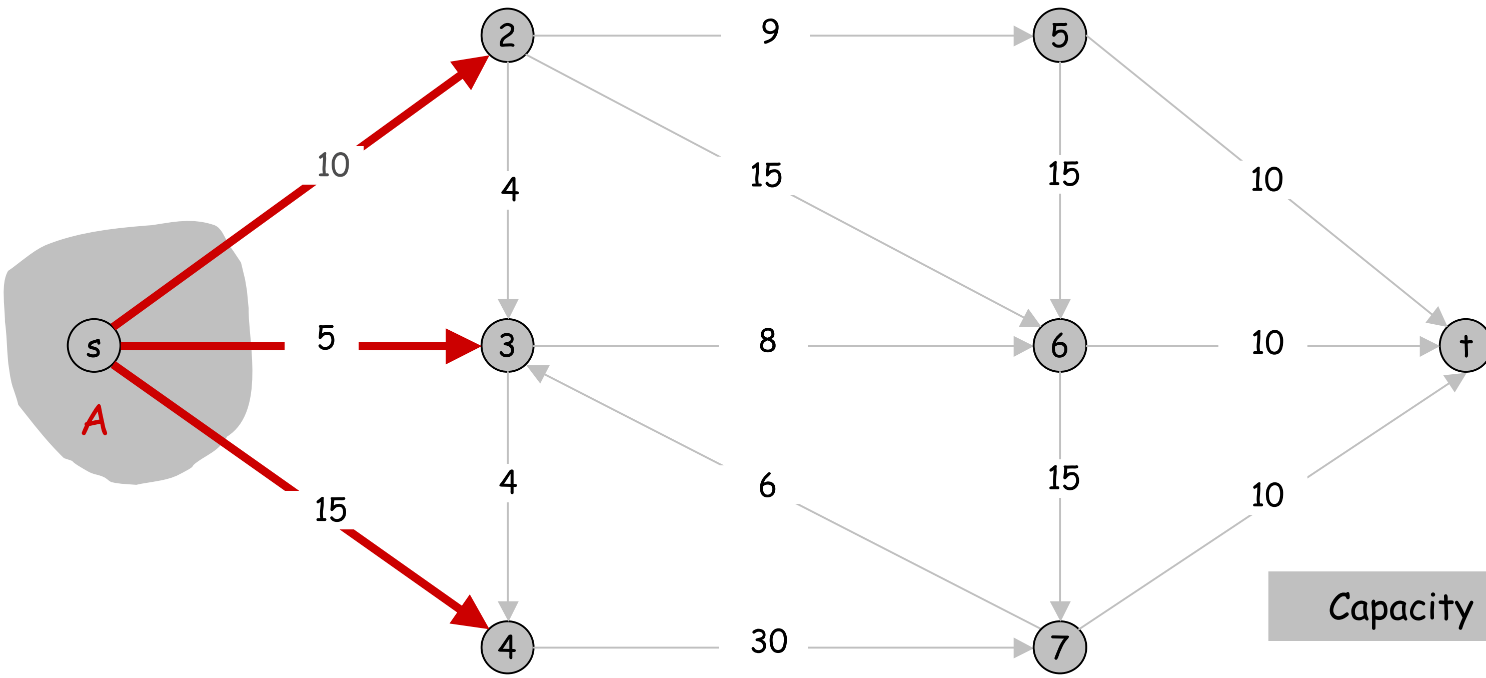


$S = \{1, 2, 4, 6\}$

$C(S) = u_{45} + u_{47} + u_{43} + u_{67}$

Cut Upperbound

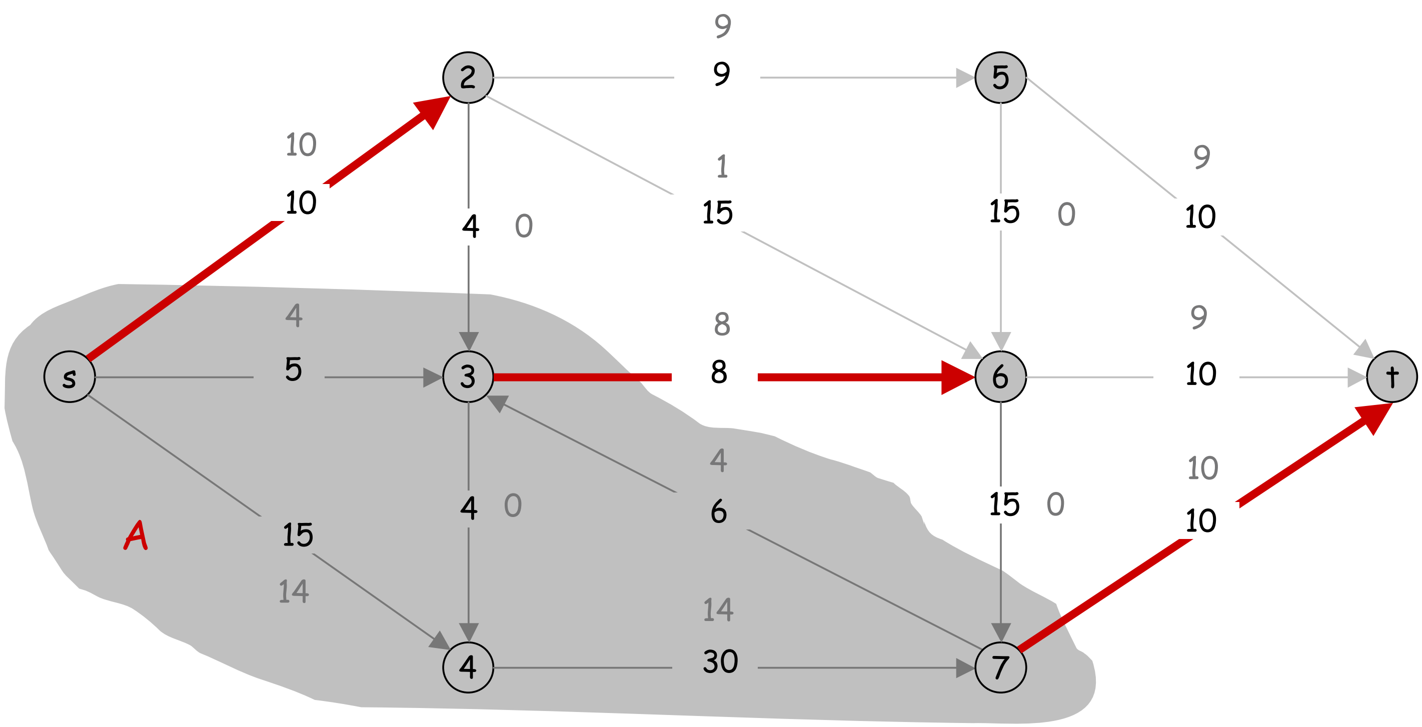
Cut capacity = 30 $\Rightarrow$ Flow value $\leq$ 30



Capacity = 30

Cut Upperbound

Value of flow = 28
Cut capacity = 28 $\Rightarrow$ Flow value ≤ 28



Ford-Fulkerson converges to the optimum

Theorem 1. *If the Ford-Fulkerson algorithm terminates because no augmenting path can be found, then the current flow is* ***optimal****.*

Proof idea:

- if no augmenting path has been found, the labeling algorithm has failed.
- Let S denote the set of nodes that were included in I at some point.
- Obviously $t \notin S$ and $s \in S$. Therefore S is a cut.
- We can show that the current flow is equal to the capacity of that cut $C(S)$ and is hence optimal.