

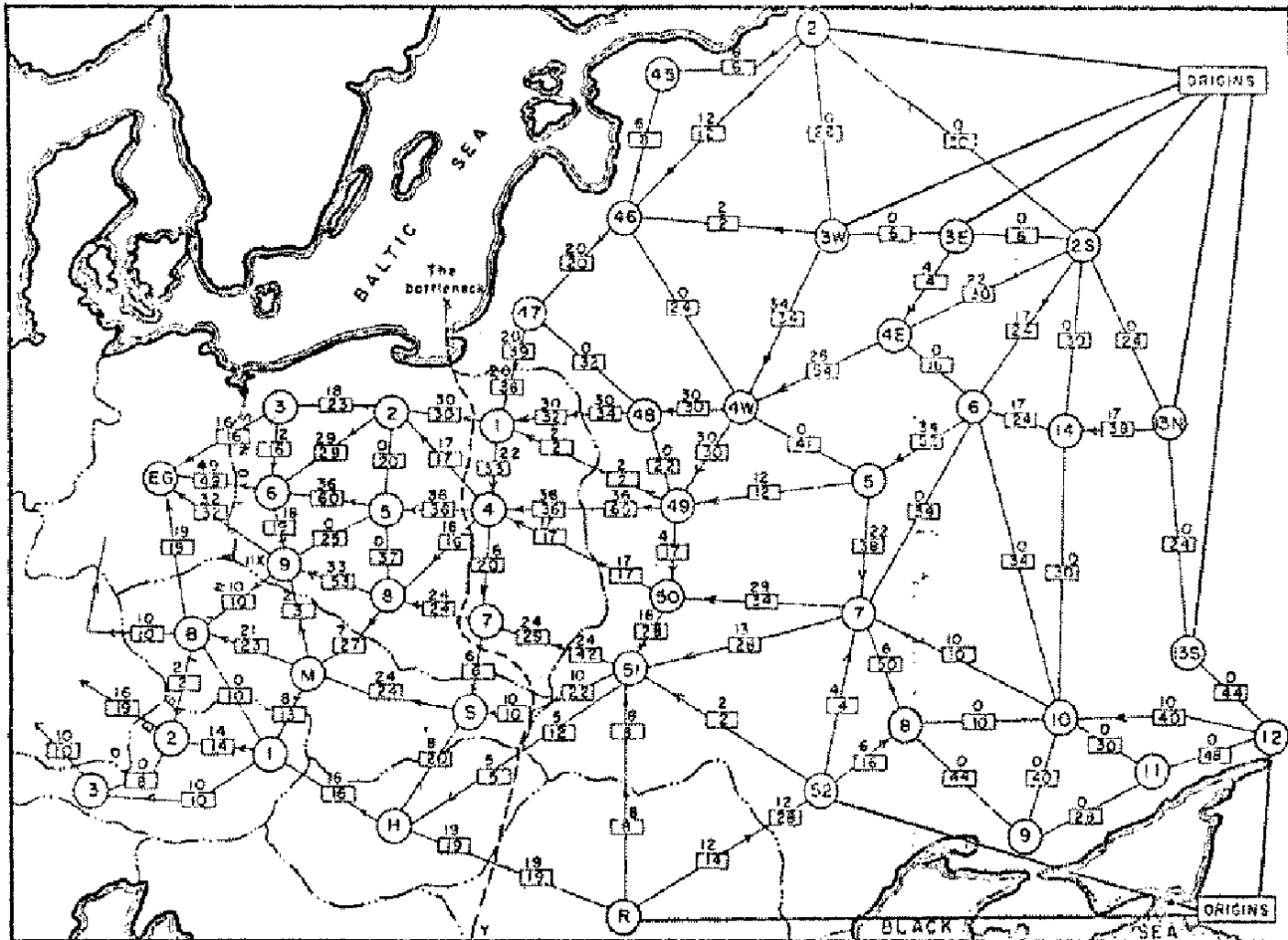
Chapter 7

Network Flow



Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

Soviet Rail Network, 1955



Reference: *On the history of the transportation and maximum flow problems.*
Alexander Schrijver in *Math Programming*, 91: 3, 2002.

Maximum Flow and Minimum Cut

Max flow and min cut.

- Two very rich algorithmic problems.
- Cornerstone problems in combinatorial optimization.
- Beautiful mathematical duality.

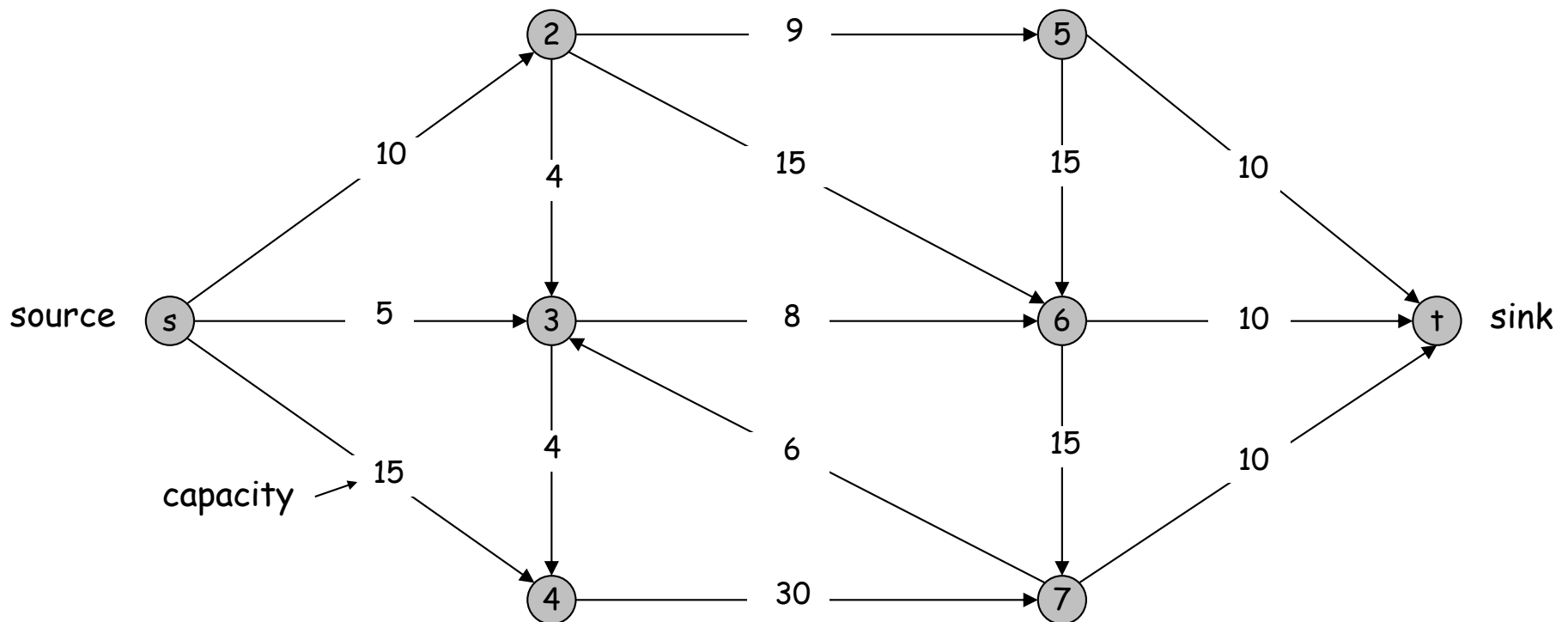
Nontrivial applications / reductions.

- Data mining.
- Open-pit mining.
- Project selection.
- Airline scheduling.
- Bipartite matching.
- Baseball elimination.
- Image segmentation.
- Network connectivity.
- Network reliability.
- Distributed computing.
- Egalitarian stable matching.
- Security of statistical data.
- Network intrusion detection.
- Multi-camera scene reconstruction.
- Many many more . . .

Minimum Cut Problem

Flow network.

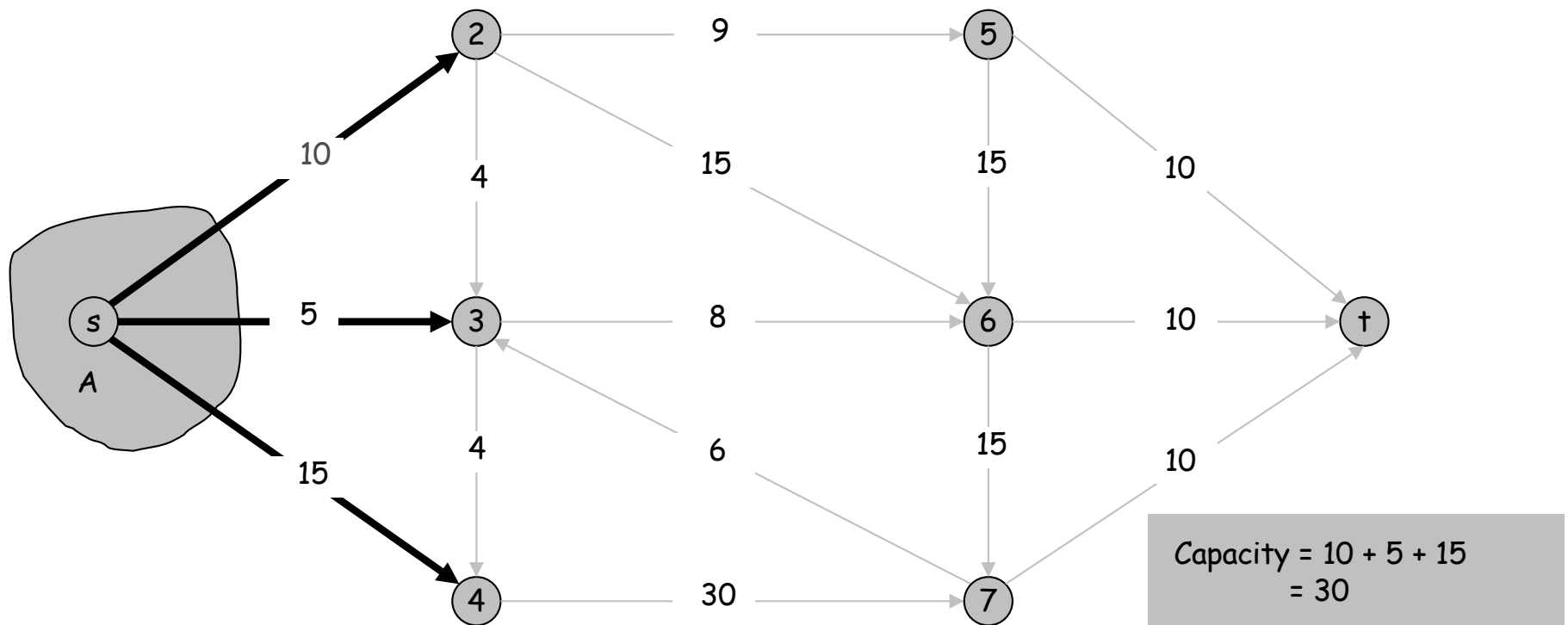
- Abstraction for material **flowing** through the edges.
- $G = (V, E)$ = directed graph, no parallel edges.
- Two distinguished nodes: s = source, t = sink.
- $c(e)$ = capacity of edge e .



Cuts

Def. An **s-t cut** is a partition (A, B) of V with $s \in A$ and $t \in B$.

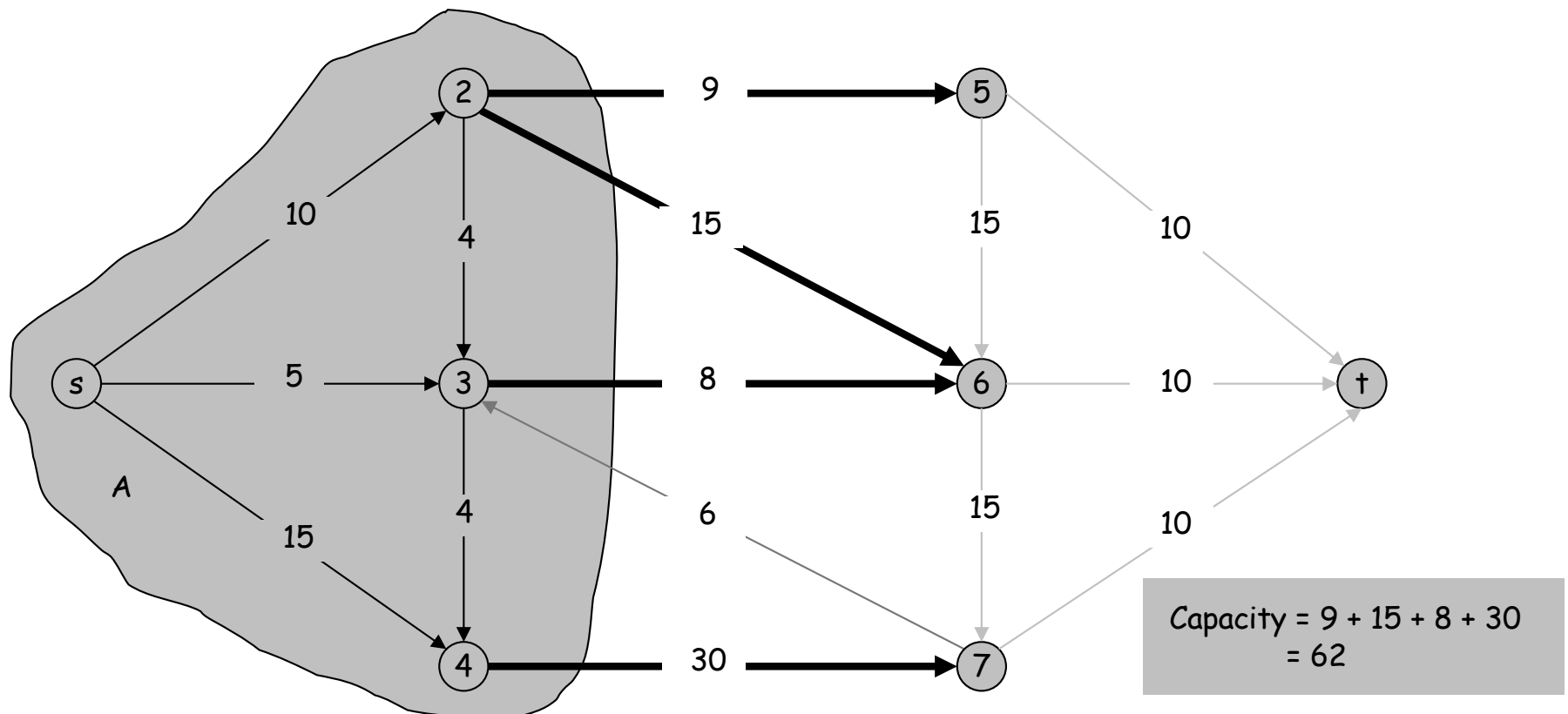
Def. The **capacity** of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Cuts

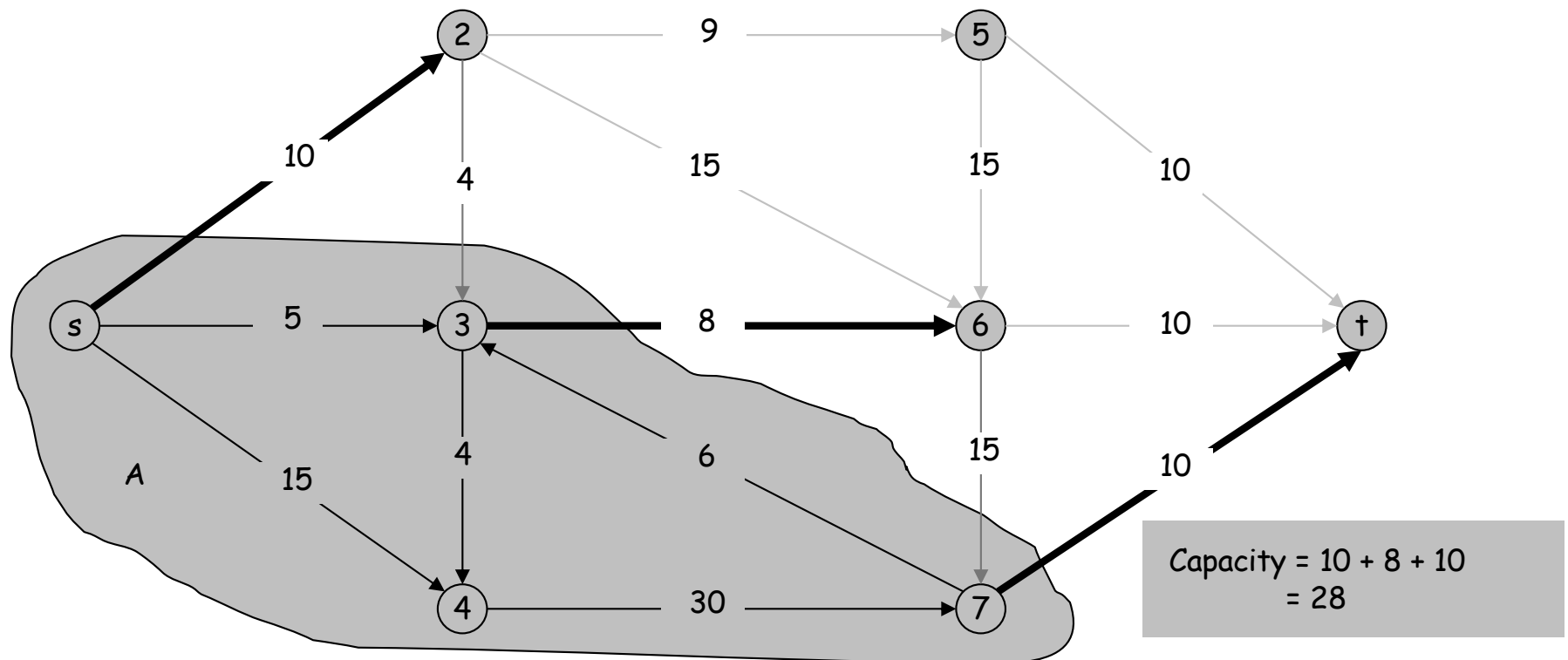
Def. An **s-t cut** is a partition (A, B) of V with $s \in A$ and $t \in B$.

Def. The **capacity** of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



Minimum Cut Problem

Min s-t cut problem. Find an s-t cut of minimum capacity.

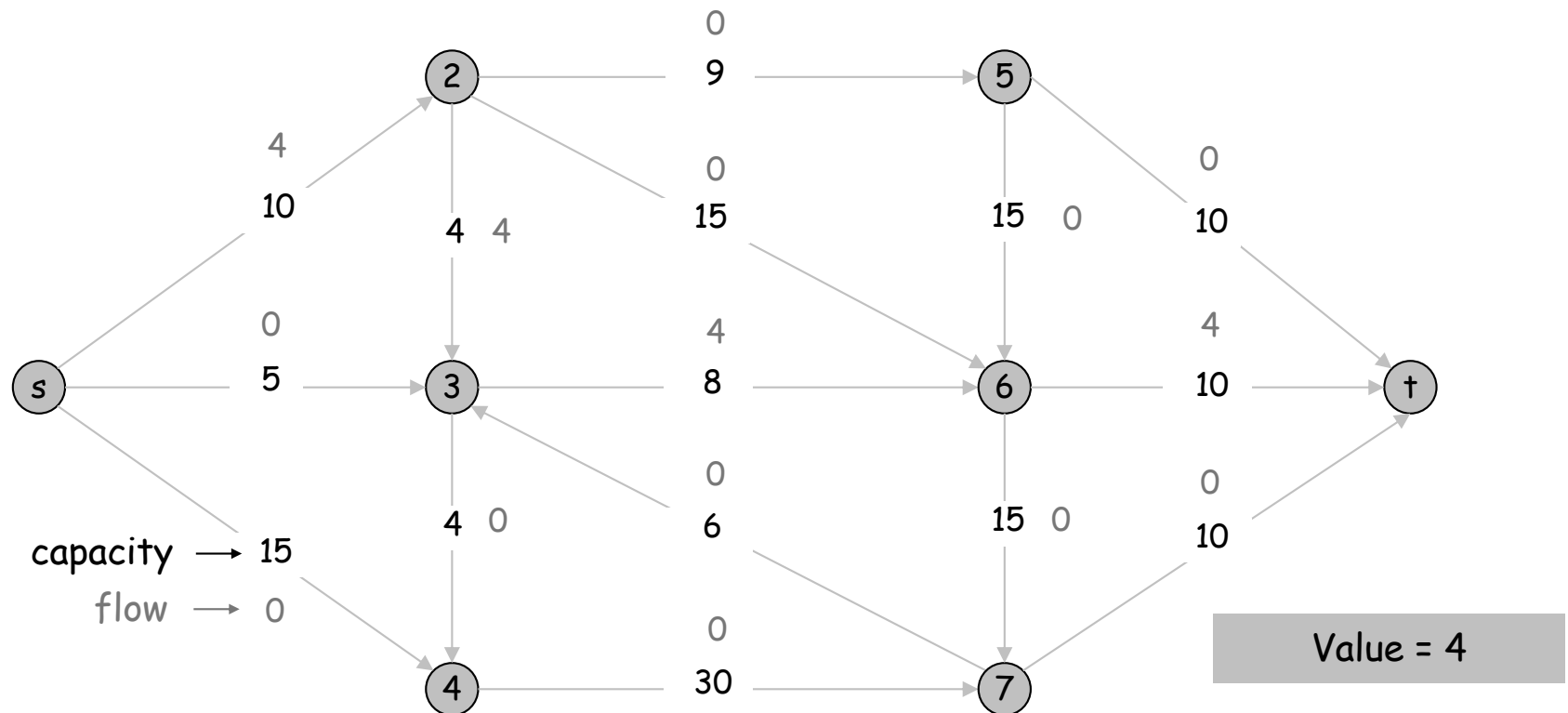


Flows

Def. An **s-t flow** is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

Def. The **value** of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.

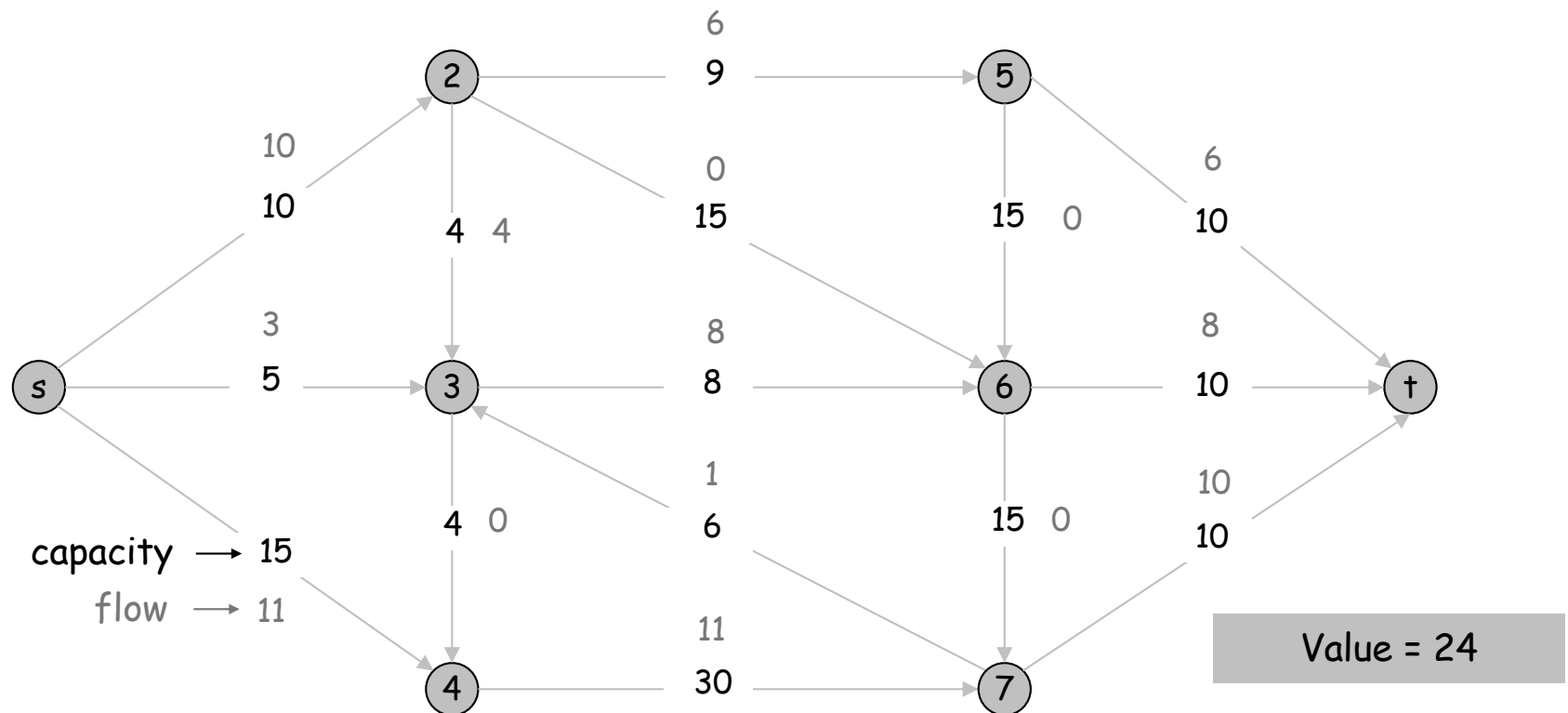


Flows

Def. An **s-t flow** is a function that satisfies:

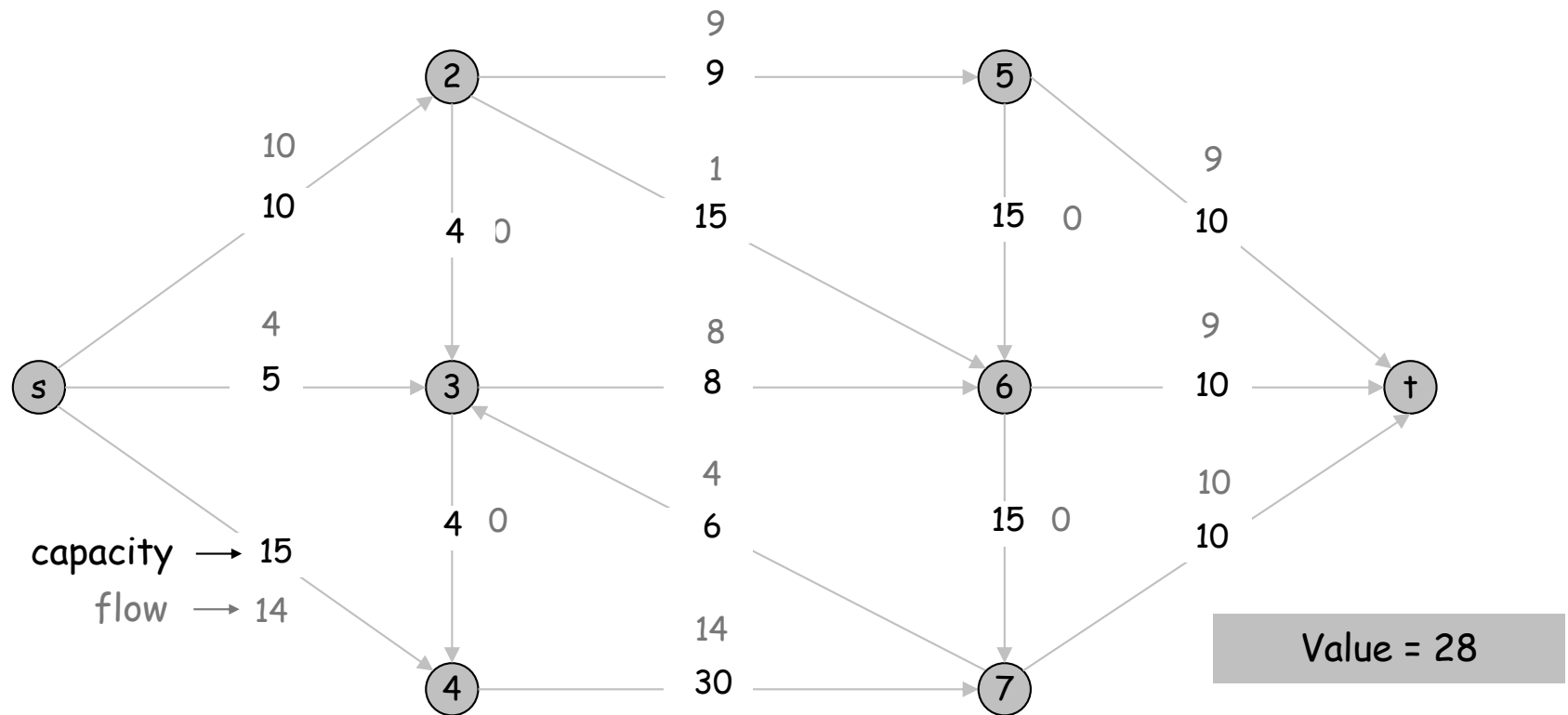
- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

Def. The **value** of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Maximum Flow Problem

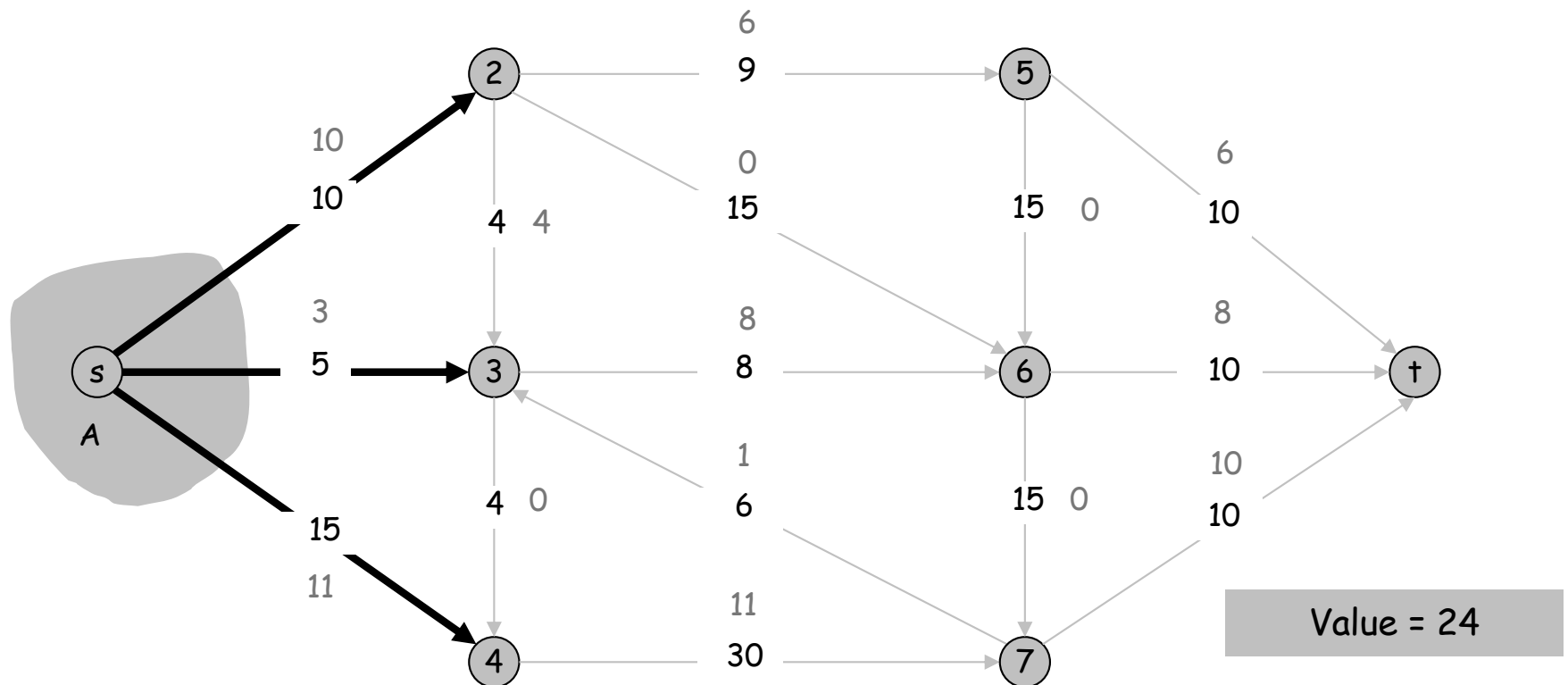
Max flow problem. Find s-t flow of maximum value.



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s - t cut. Then, the net flow sent across the cut is equal to the amount leaving s .

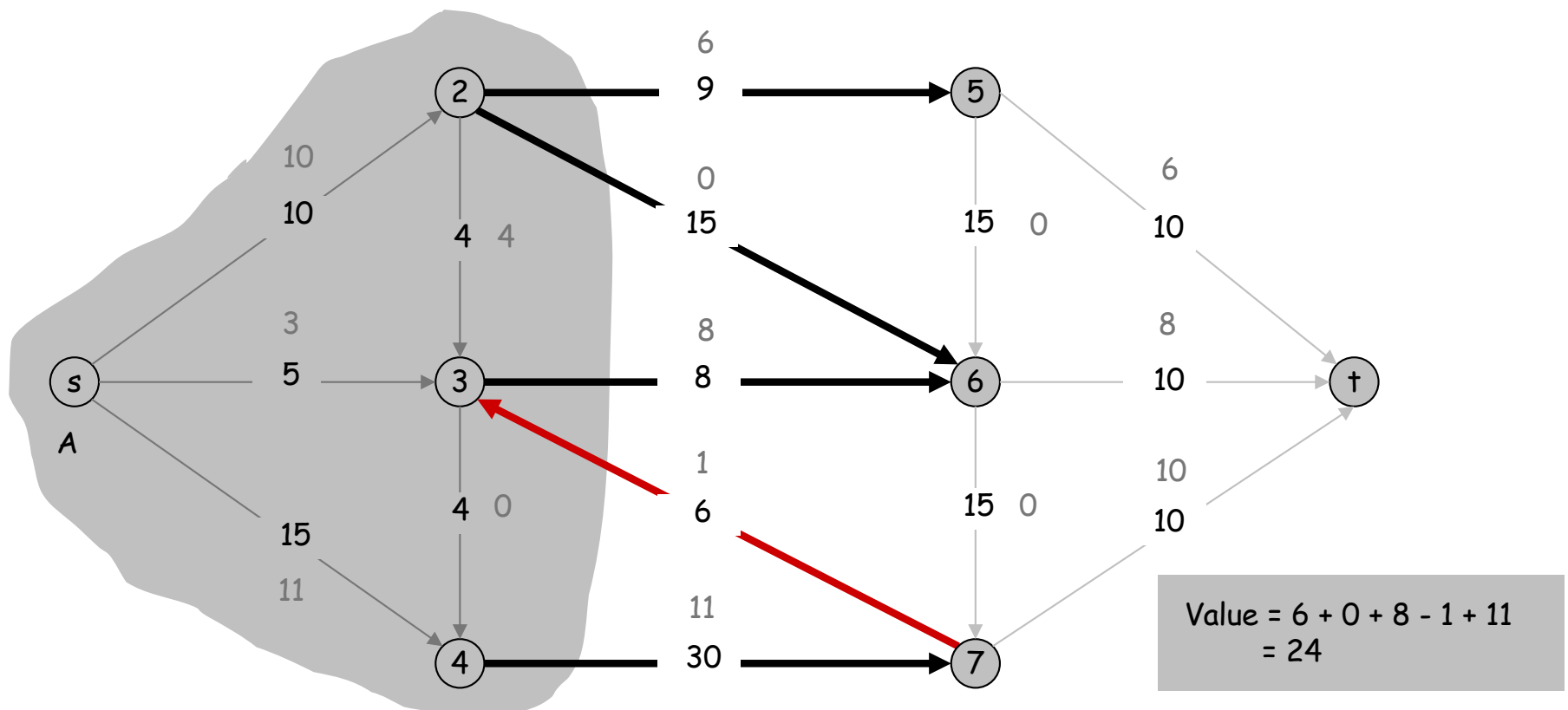
$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s - t cut. Then, the net flow sent across the cut is equal to the amount leaving s .

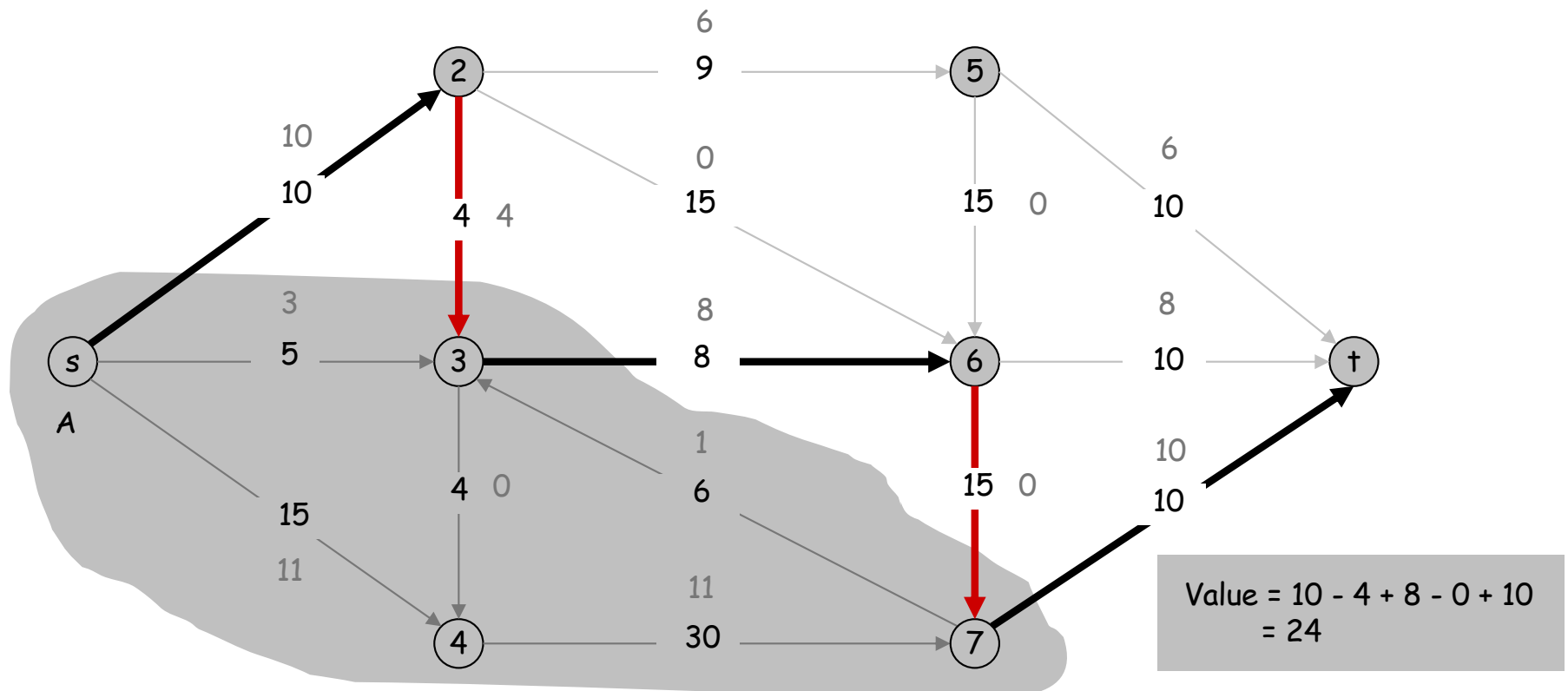
$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s - t cut. Then, the net flow sent across the cut is equal to the amount leaving s .

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f)$$



Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s - t cut. Then

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f).$$

Pf.

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

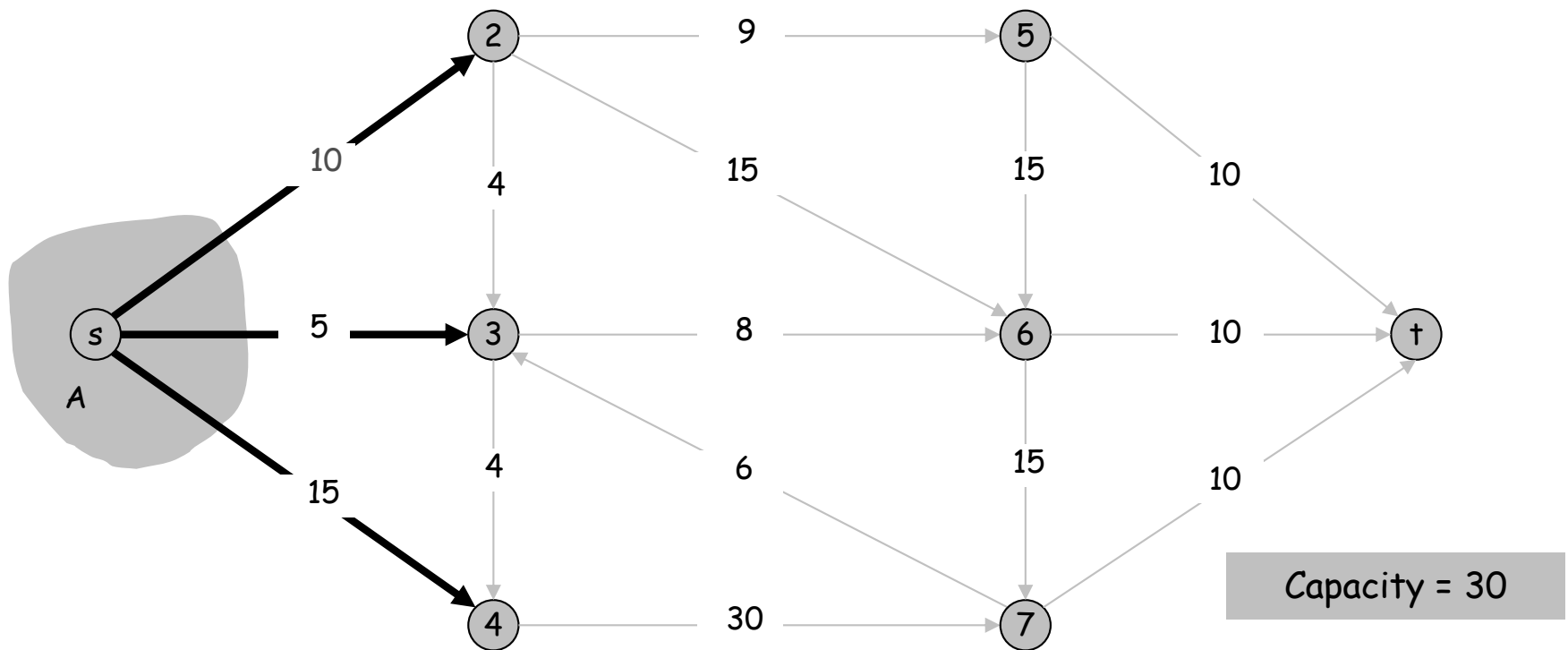
$$\begin{array}{l} \text{by flow conservation, all terms} \\ \text{except } v = s \text{ are 0} \end{array} \quad \longrightarrow \quad = \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)$$

$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e).$$

Flows and Cuts

Weak duality. Let f be any flow, and let (A, B) be any s - t cut. Then the value of the flow is at most the capacity of the cut.

Cut capacity = 30 \Rightarrow Flow value ≤ 30

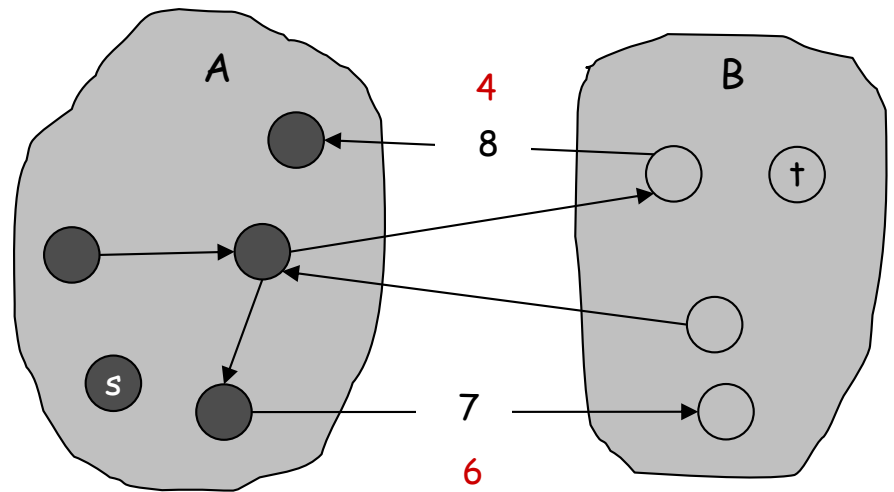


Flows and Cuts

Weak duality. Let f be any flow. Then, for any s - t cut (A, B) we have $v(f) \leq \text{cap}(A, B)$.

Pf.

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\leq \sum_{e \text{ out of } A} c(e) \\ &\leq \sum_{e \text{ out of } A} c(e) \\ &= \text{cap}(A, B) \quad \blacksquare \end{aligned}$$

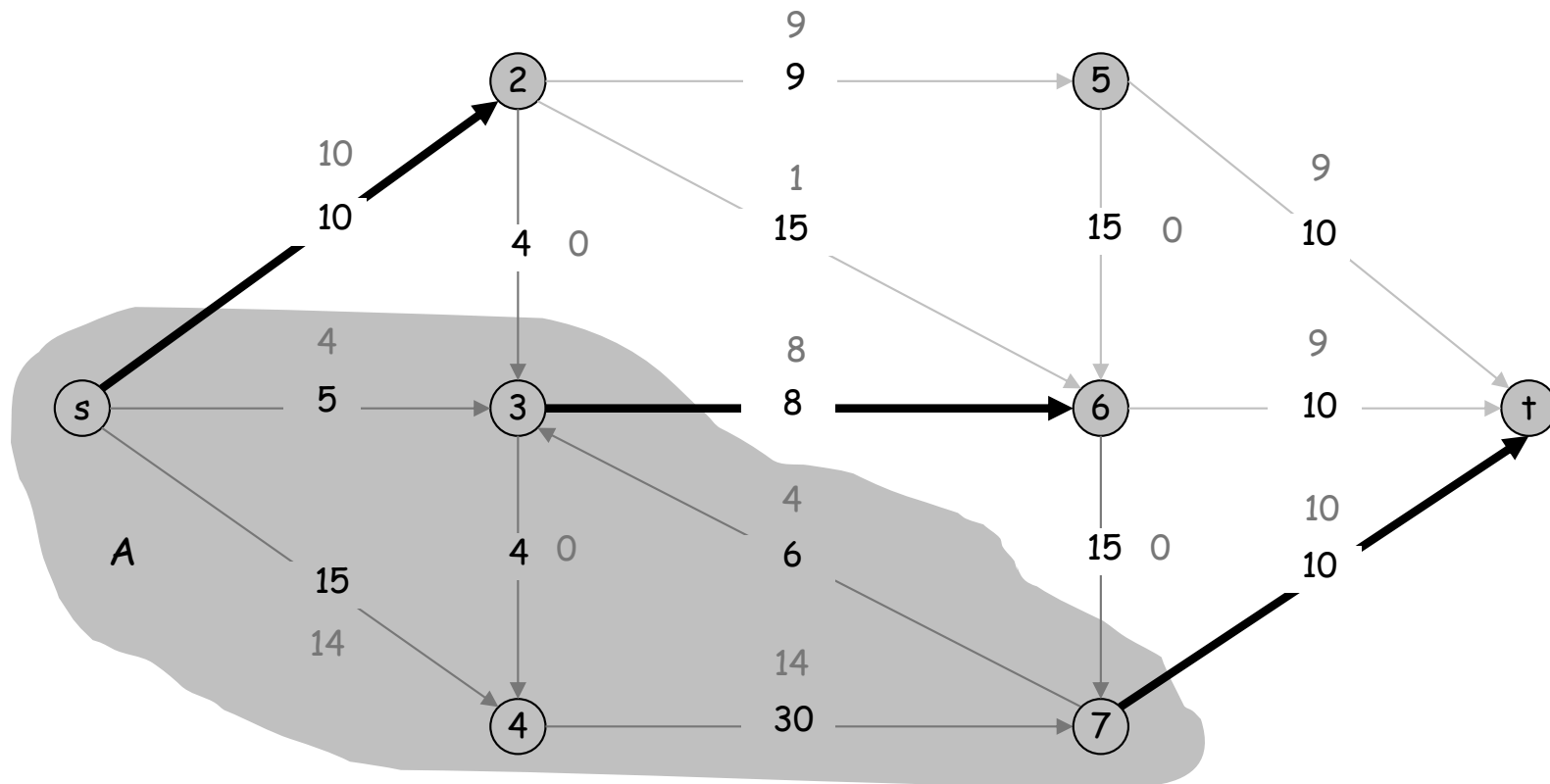


Certificate of Optimality

Corollary. Let f be any flow, and let (A, B) be any cut. If $v(f) = \text{cap}(A, B)$, then f is a max flow and (A, B) is a min cut.

Value of flow = 28

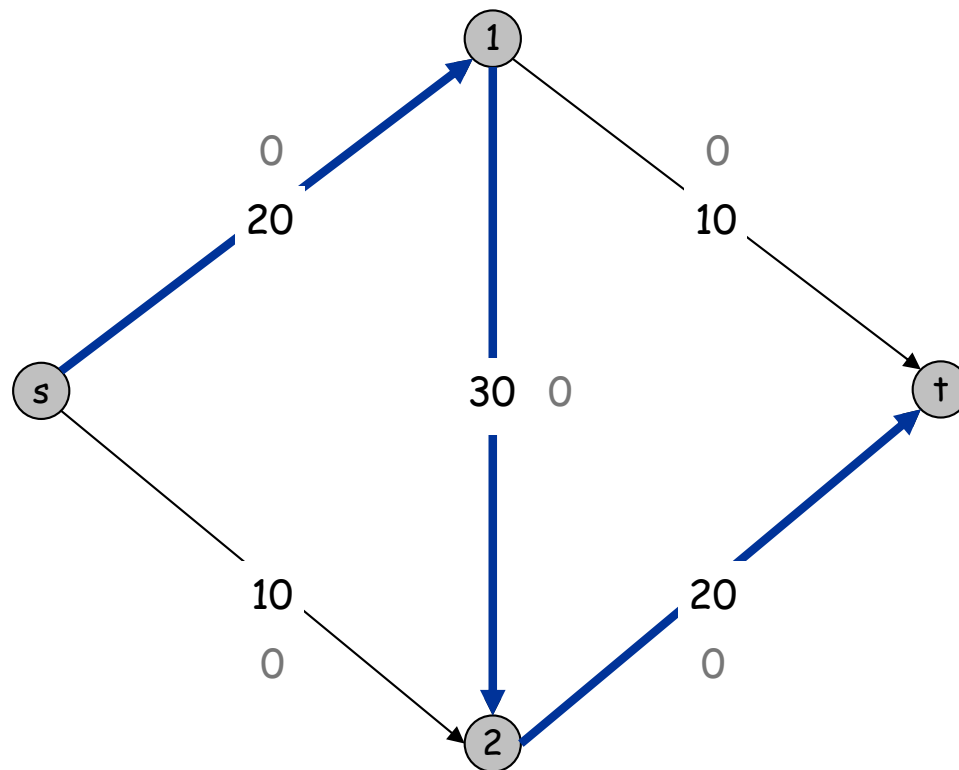
Cut capacity = 28 \Rightarrow Flow value ≤ 28



Towards a Max Flow Algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s - t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P .
- Repeat until you get stuck.

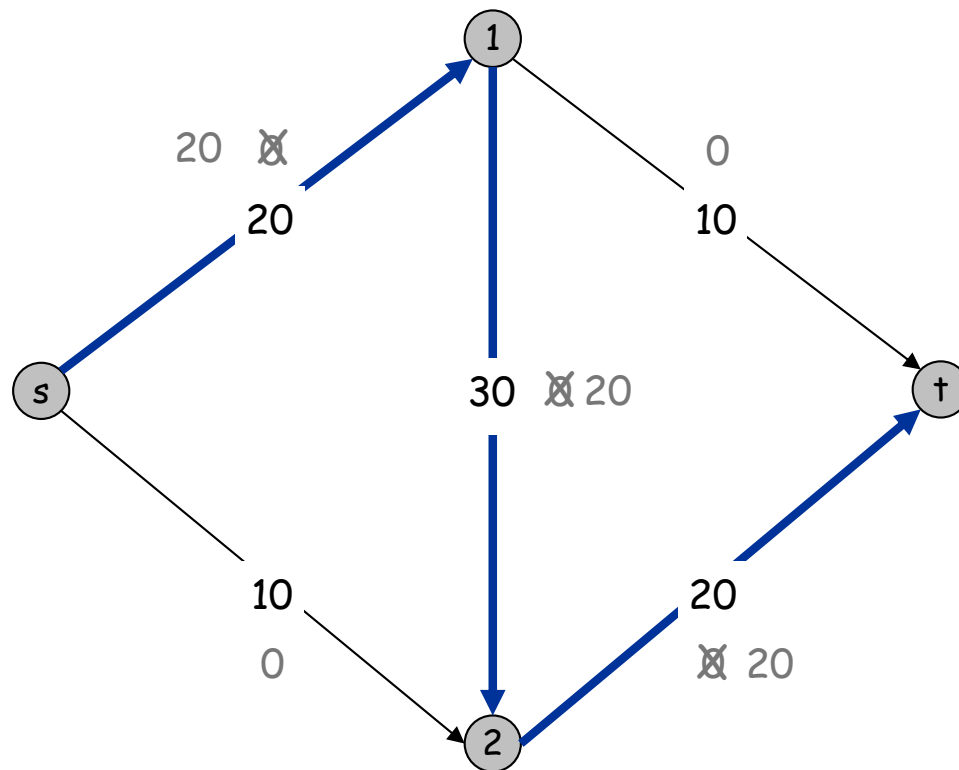


Flow value = 0

Towards a Max Flow Algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s - t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P .
- Repeat until you get stuck.



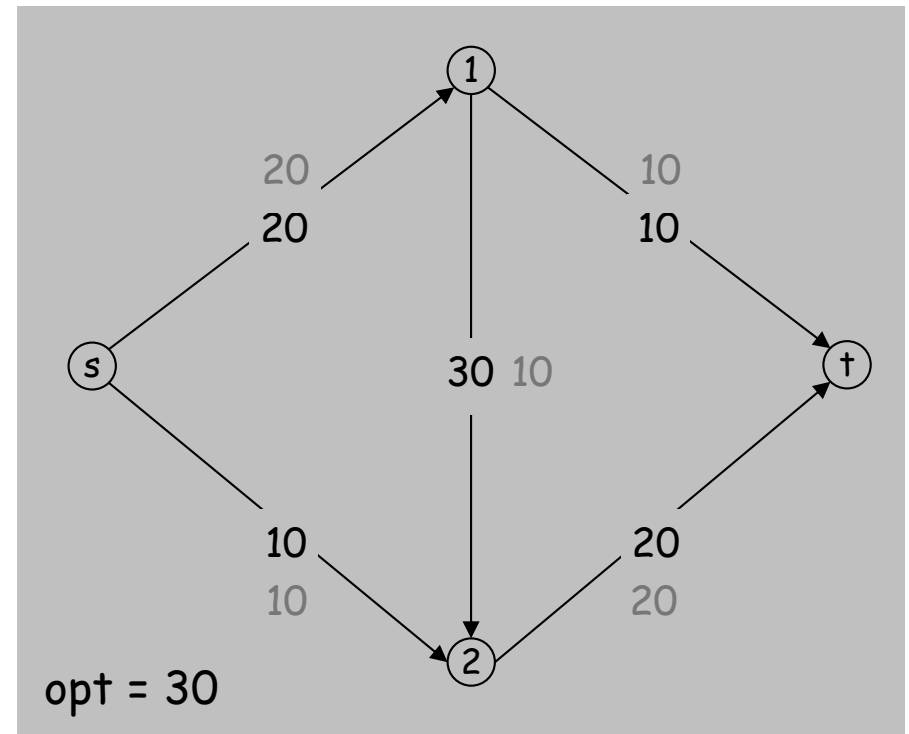
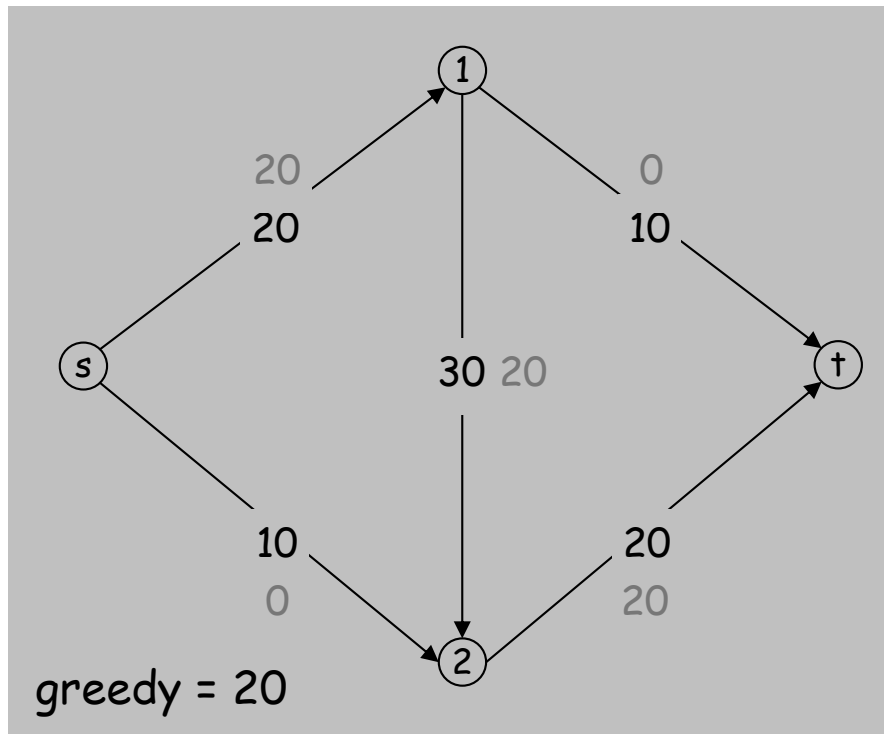
Flow value = 20

Towards a Max Flow Algorithm

Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s - t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P .
- Repeat until you get **stuck**.

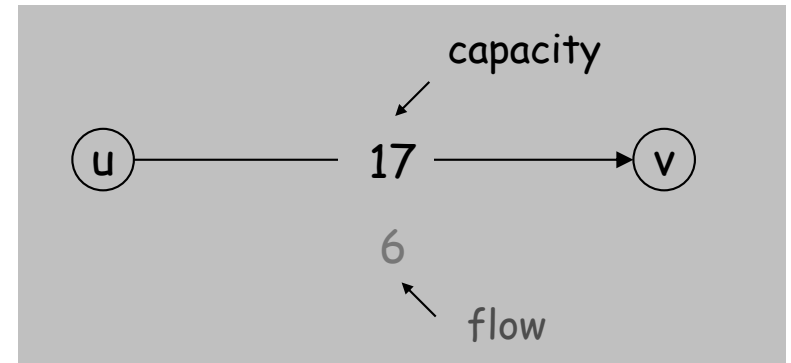
↖ locally optimality \nRightarrow global optimality



Residual Graph

Original edge: $e = (u, v) \in E$.

- Flow $f(e)$, capacity $c(e)$.

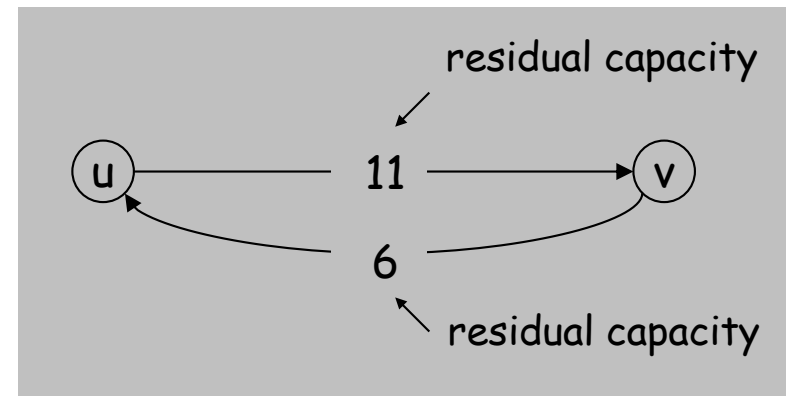


Residual edge.

- "Undo" flow sent.
- $e = (u, v)$ and $e^R = (v, u)$.
- Residual capacity: for e in E

$$c_f(e) = c(e) - f(e)$$

$$c_f(e^R) = f(e)$$

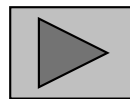
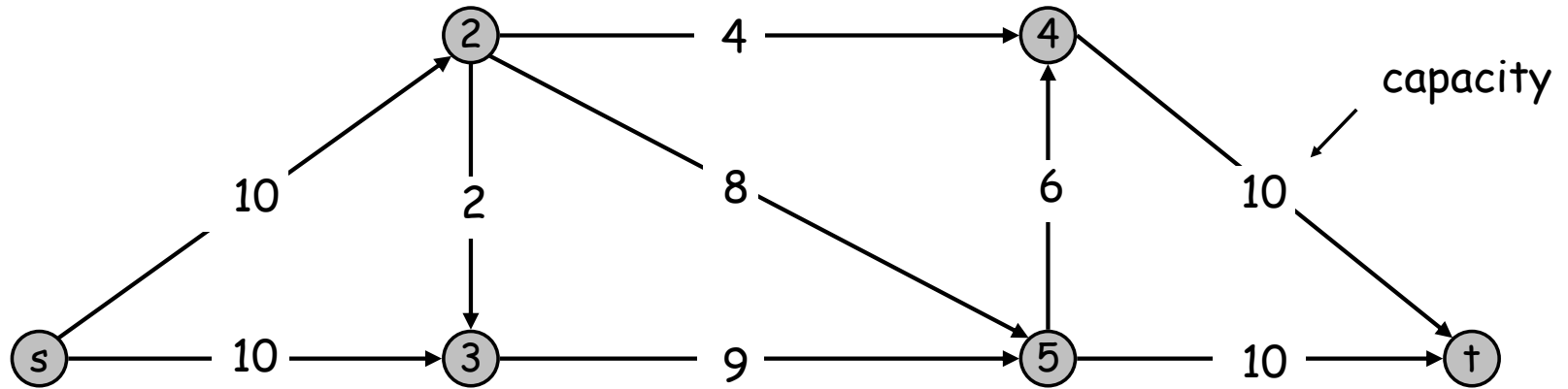


Residual graph: $G_f = (V, E_f)$.

- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

Ford-Fulkerson Algorithm

G :



Augmenting Path Algorithm

```
Augment(f, c, P) {  
    b ← bottleneck(P)  
    foreach (u,v) ∈ P {  
        if ((u,v) ∈ E)  
            f((u,v)) ← f((u,v)) + b  
        else  
            f((v,u)) ← f((v,u)) - b  
    }  
    return f  
}
```

forward edge

reverse edge

```
Ford-Fulkerson(G, s, t, c) {  
    foreach e ∈ E f(e) ← 0  
    Gf ← residual graph  
  
    while (there exists augmenting path P) {  
        f ← Augment(f, c, P)  
        update Gf  
    }  
    return f  
}
```

Max-Flow Min-Cut Theorem

Augmenting path theorem. Flow f is a max flow iff there are no augmenting paths.

Max-flow min-cut theorem. [Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut.

Proof strategy. We prove both simultaneously by showing that TFAE:

- (i) There exists a cut (A, B) such that $v(f) = \text{cap}(A, B)$.
- (ii) Flow f is a max flow.
- (iii) There is no augmenting path relative to f .

(i) \Rightarrow (ii) This was the corollary to weak duality lemma.

(ii) \Rightarrow (iii) We show contrapositive.

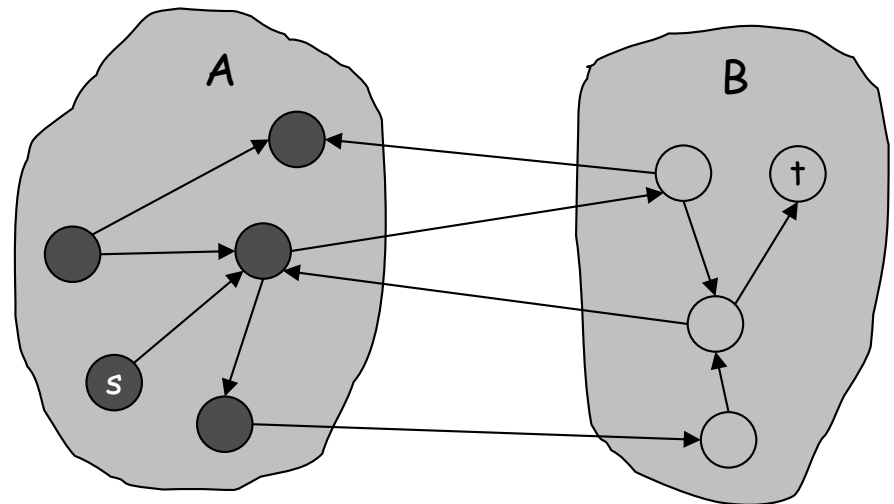
- Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.

Proof of Max-Flow Min-Cut Theorem

(iii) \Rightarrow (i)

- Let f be a flow with no augmenting paths.
- Let A be set of vertices reachable from s in residual graph.
- By definition of A , $s \in A$.
- By definition of f , $t \notin A$.

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &= \sum_{e \text{ out of } A} c(e) \\ &= \text{cap}(A, B) \quad \blacksquare \end{aligned}$$



original network

Running Time

Assumption. All capacities are integers between 1 and C .

Invariant. Every flow value $f(e)$ and every residual capacities $c_f(e)$ remains an integer throughout the algorithm.

Theorem. The algorithm terminates in at most $v(f^*) \leq nC$ iterations.

Pf. Each augmentation increase value by at least 1. ■

Corollary. If $C = 1$, Ford-Fulkerson runs in $O(mn)$ time.


Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value $f(e)$ is an integer.

Pf. Since algorithm terminates, theorem follows from invariant. ■

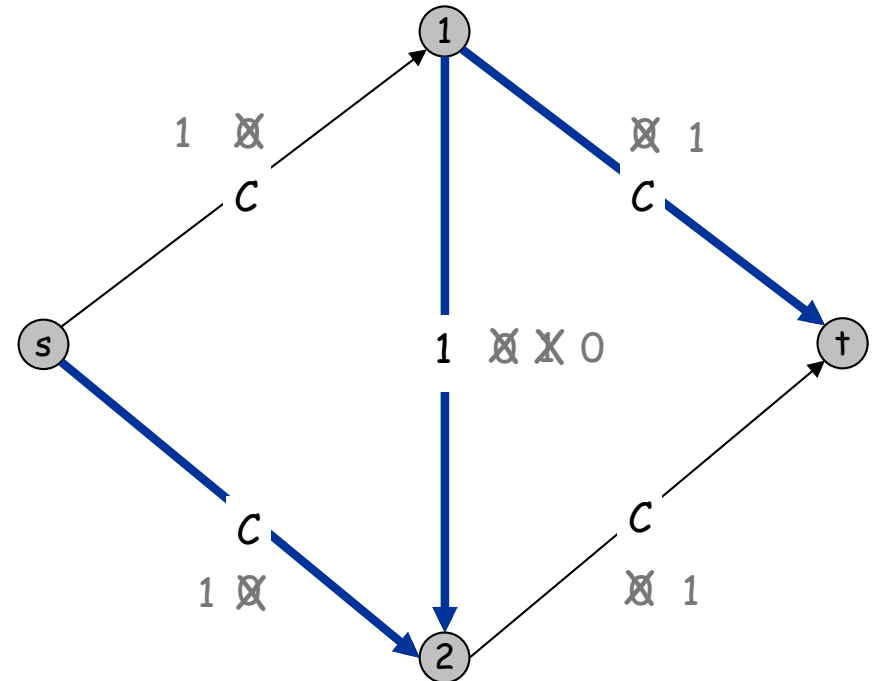
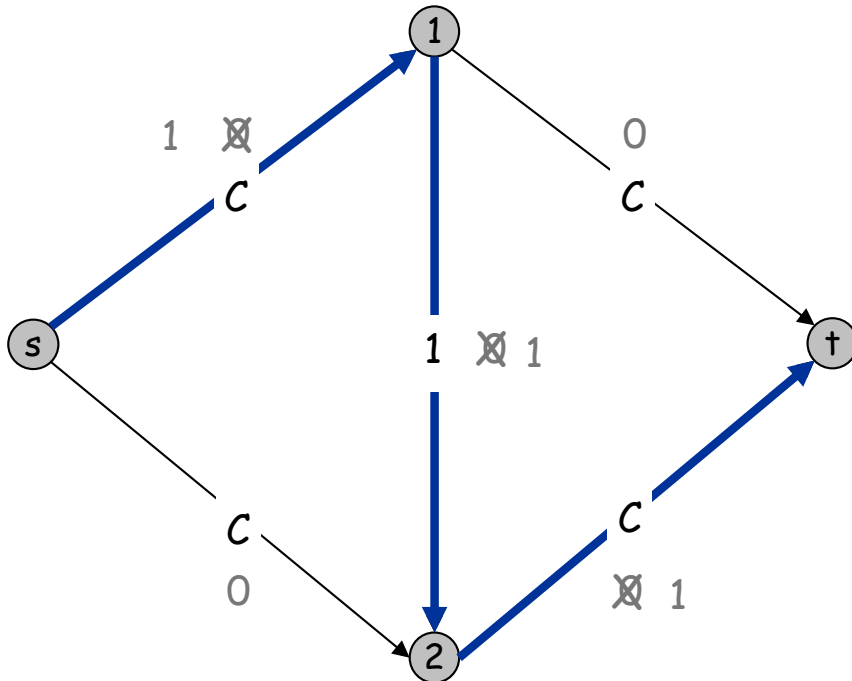
7.3 Choosing Good Augmenting Paths

Ford-Fulkerson: Exponential Number of Augmentations

Q. Is generic Ford-Fulkerson algorithm polynomial in input size?

m , n , and $\log C$ 

A. No. If max capacity is C , then algorithm can take C iterations.



Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

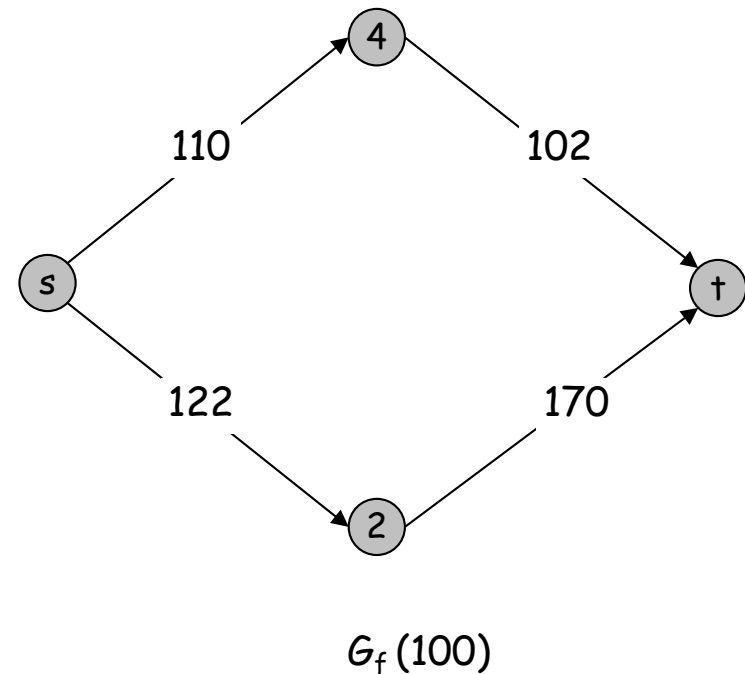
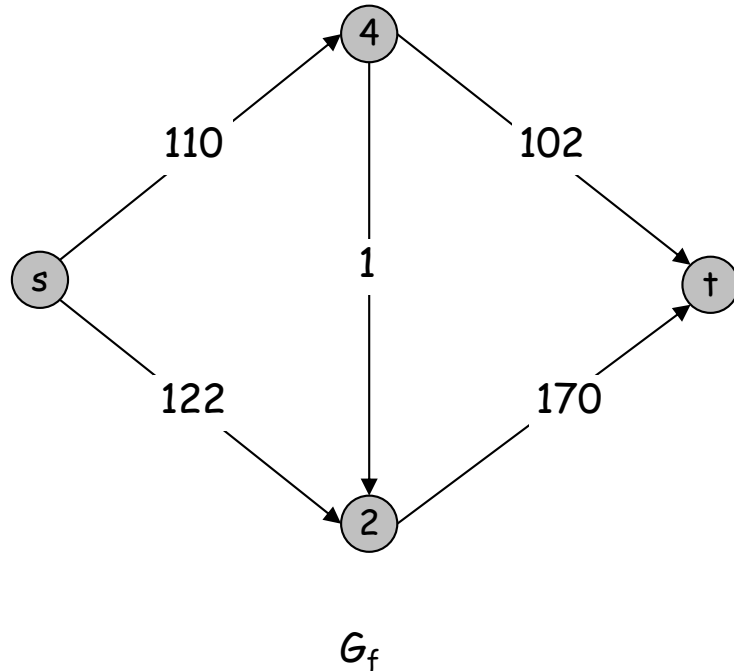
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ .
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ .



Capacity Scaling

```
Scaling-Max-Flow( $G, s, t, c$ ) {  
  foreach  $e \in E$   $f(e) \leftarrow 0$   
   $\Delta \leftarrow$  smallest power of 2 greater than or equal to  $C$   
   $G_f \leftarrow$  residual graph  
  
  while ( $\Delta \geq 1$ ) {  
     $G_f(\Delta) \leftarrow \Delta$ -residual graph  
    while (there exists augmenting path  $P$  in  $G_f(\Delta)$ ) {  
       $f \leftarrow$  augment( $f, c, P$ )  
      update  $G_f(\Delta)$   
    }  
     $\Delta \leftarrow \Delta / 2$   
  }  
  return  $f$   
}
```

Capacity Scaling: Correctness

Assumption. All edge capacities are integers between 1 and C .

Integrality invariant. All flow and residual capacity values are integral.

Correctness. If the algorithm terminates, then f is a max flow.

Pf.

- By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths. ▪

Capacity Scaling: Running Time

Lemma 1. The outer while loop repeats $1 + \lceil \log_2 C \rceil$ times.

Pf. Initially $C \leq \Delta < 2C$. Δ decreases by a factor of 2 each iteration. ■

Lemma 2. Let f be the flow at the end of a Δ -scaling phase. Then the value of the maximum flow is at most $v(f) + m \Delta$. ← proof on next slide

Lemma 3. There are at most $2m$ augmentations per scaling phase.

- Let f be the flow at the end of the previous scaling phase (for 2Δ).
- $L2 \Rightarrow v(f^*) \leq v(f) + m (2\Delta)$.
- Each augmentation in a Δ -phase increases $v(f)$ by at least Δ . ■

Theorem. The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. It can be implemented to run in $O(m^2 \log C)$ time. ■

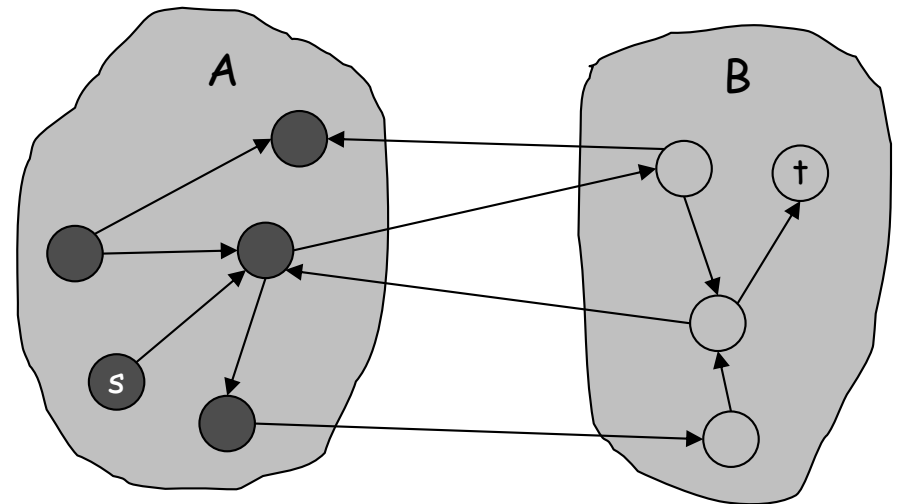
Capacity Scaling: Running Time

Lemma 2. Let f be the flow at the end of a Δ -scaling phase. Then value of the maximum flow is at most $v(f) + m \Delta$.

Pf. (almost identical to proof of max-flow min-cut theorem)

- We show that at the end of a Δ -phase, there exists a cut (A, B) such that $\text{cap}(A, B) \leq v(f) + m \Delta$.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of A , $s \in A$.
- By definition of f , $t \notin A$.

$$\begin{aligned}
 v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\
 &\geq \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\
 &= \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\
 &\geq \text{cap}(A, B) - m\Delta \quad \blacksquare
 \end{aligned}$$



original network

Fewest Edges

Edmonds-Karp/Dinitz algorithm

Augment along shortest augmenting paths (in terms of number of edges): use BFS

Running time: $O(nm^2)$; $O(mn)$ augmentations, $O(m)$ per augmentation

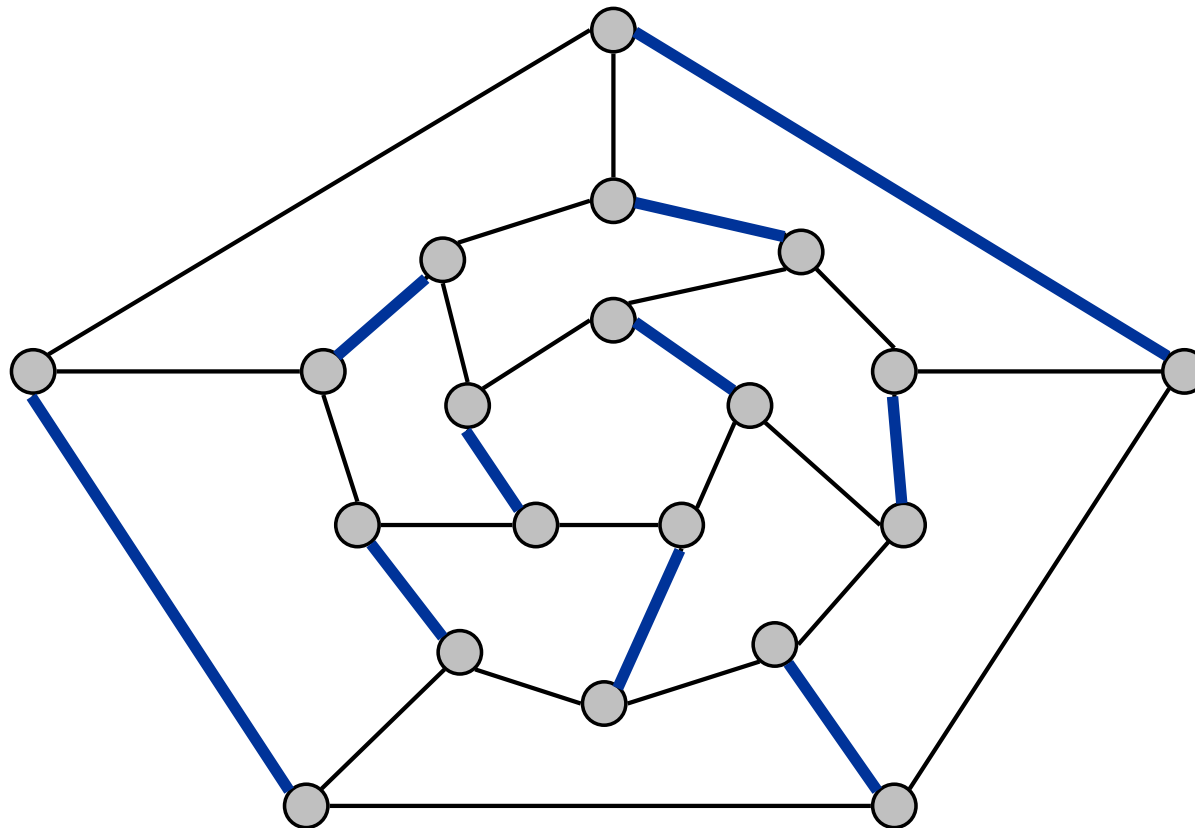
Strongly polynomial

7.5 Bipartite Matching

Matching

Matching.

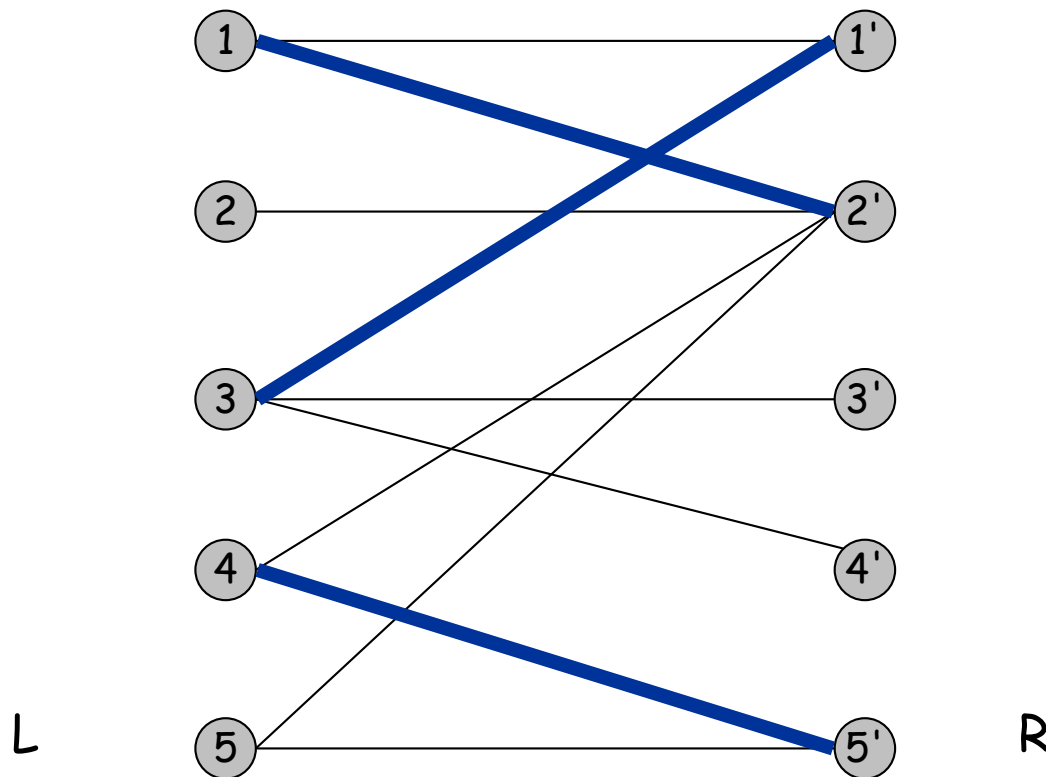
- Input: undirected graph $G = (V, E)$.
- $M \subseteq E$ is a **matching** if each node appears in at most edge in M .
- Max matching: find a max cardinality matching.



Bipartite Matching

Bipartite matching.

- Input: undirected, **bipartite** graph $G = (L \cup R, E)$.
- $M \subseteq E$ is a **matching** if each node appears in at most edge in M .
- Max matching: find a max cardinality matching.

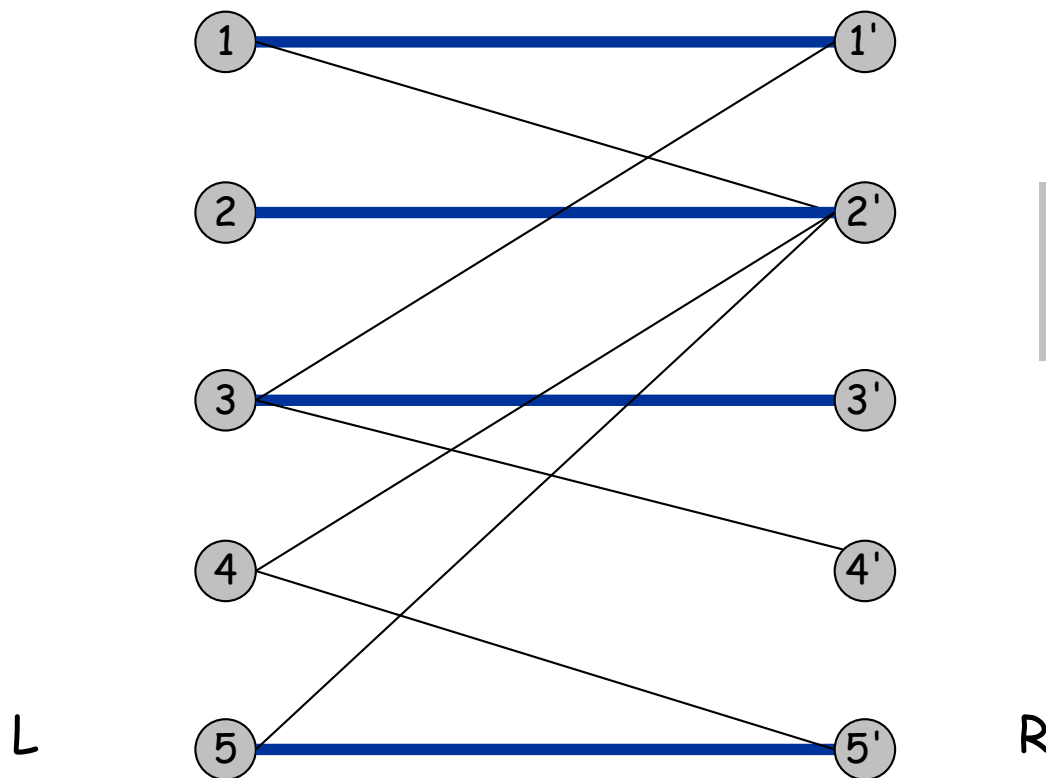


matching
1-2', 3-1', 4-5'

Bipartite Matching

Bipartite matching.

- Input: undirected, **bipartite** graph $G = (L \cup R, E)$.
- $M \subseteq E$ is a **matching** if each node appears in at most edge in M .
- Max matching: find a max cardinality matching.

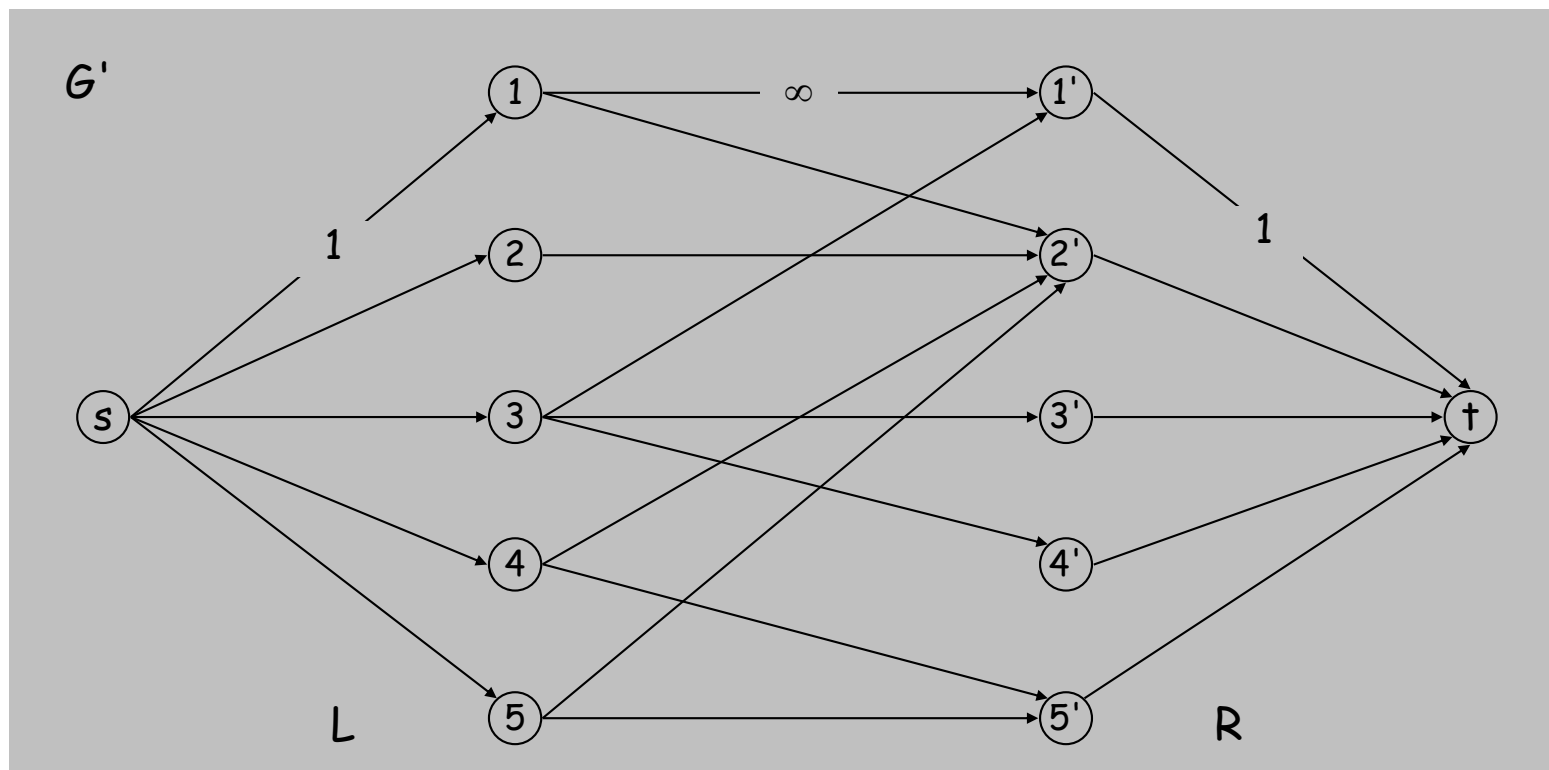


max matching
1-1', 2-2', 3-3' 4-4'

Bipartite Matching

Max flow formulation.

- Create digraph $G' = (L \cup R \cup \{s, t\}, E')$.
- Direct all edges from L to R , and assign infinite (or unit) capacity.
- Add source s , and unit capacity edges from s to each node in L .
- Add sink t , and unit capacity edges from each node in R to t .

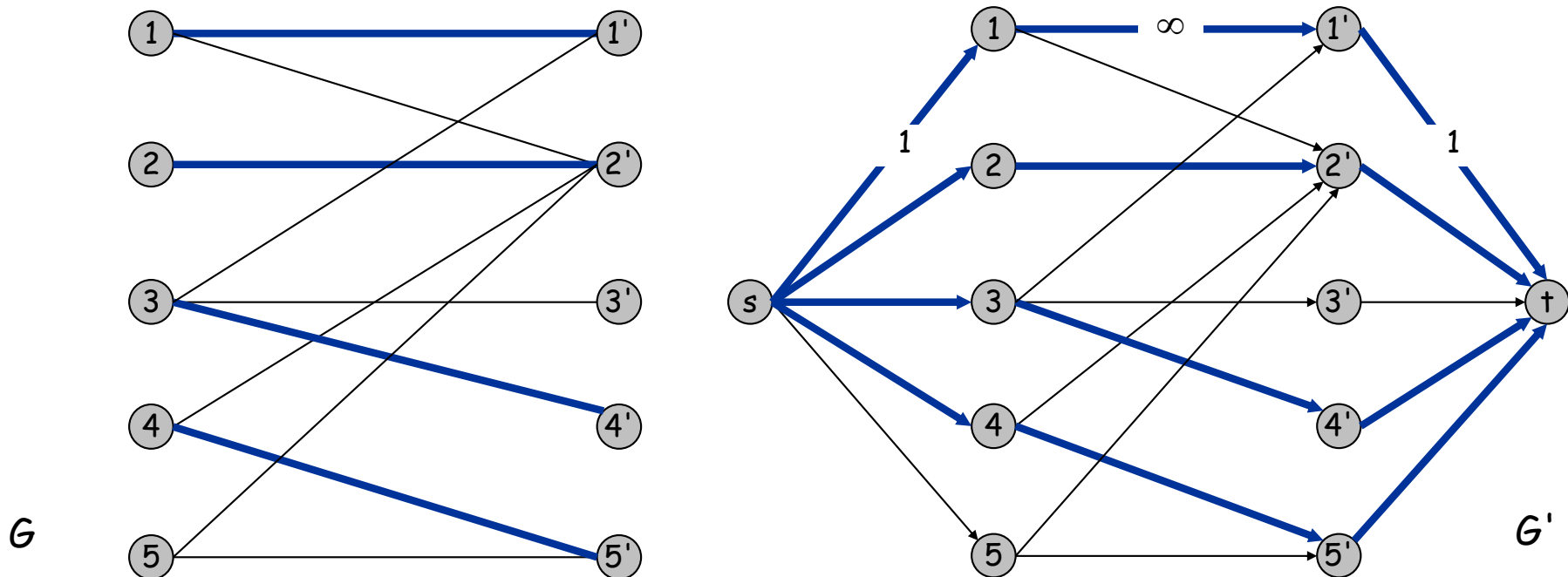


Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in G = value of max flow in G' .

Pf. \leq

- Given max matching M of cardinality k .
- Consider flow f that sends 1 unit along each of k paths.
- f is a flow, and has cardinality k . ▪

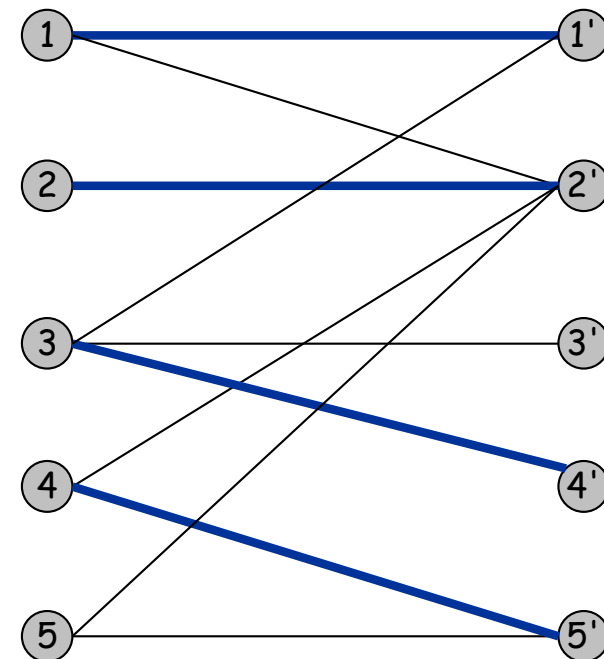
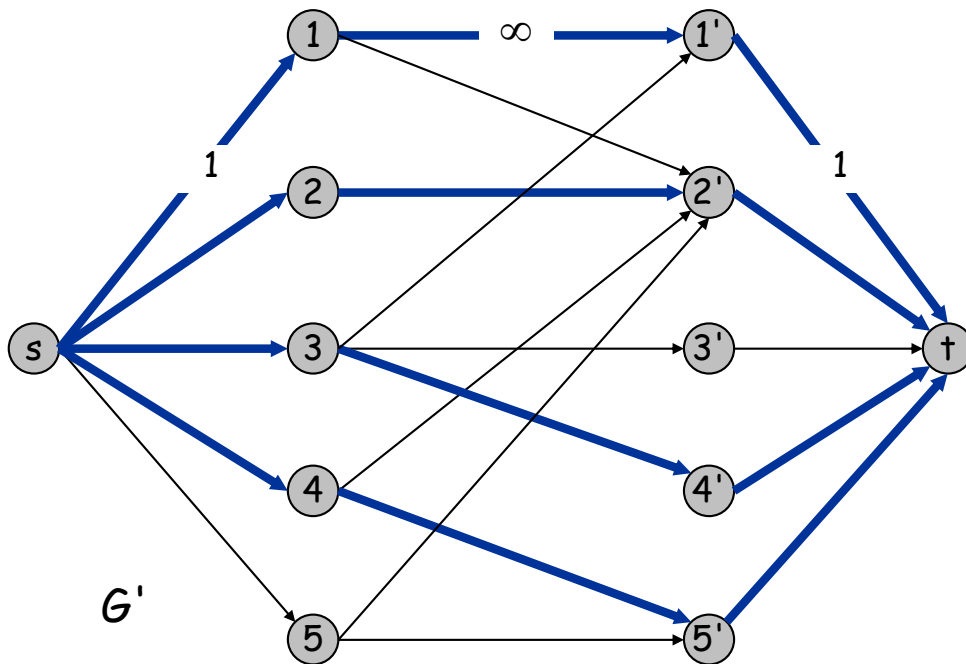


Bipartite Matching: Proof of Correctness

Theorem. Max cardinality matching in G = value of max flow in G' .

Pf. \geq

- Let f be a max flow in G' of value k .
- Integrality theorem $\Rightarrow k$ is integral and can assume f is 0-1.
- Consider M = set of edges from L to R with $f(e) = 1$.
 - each node in L and R participates in at most one edge in M
 - $|M| = k$: consider cut $(L \cup s, R \cup t)$ ▪



Perfect Matching

Def. A matching $M \subseteq E$ is **perfect** if each node appears in exactly one edge in M .

Q. When does a bipartite graph have a perfect matching?

Structure of bipartite graphs with perfect matchings.

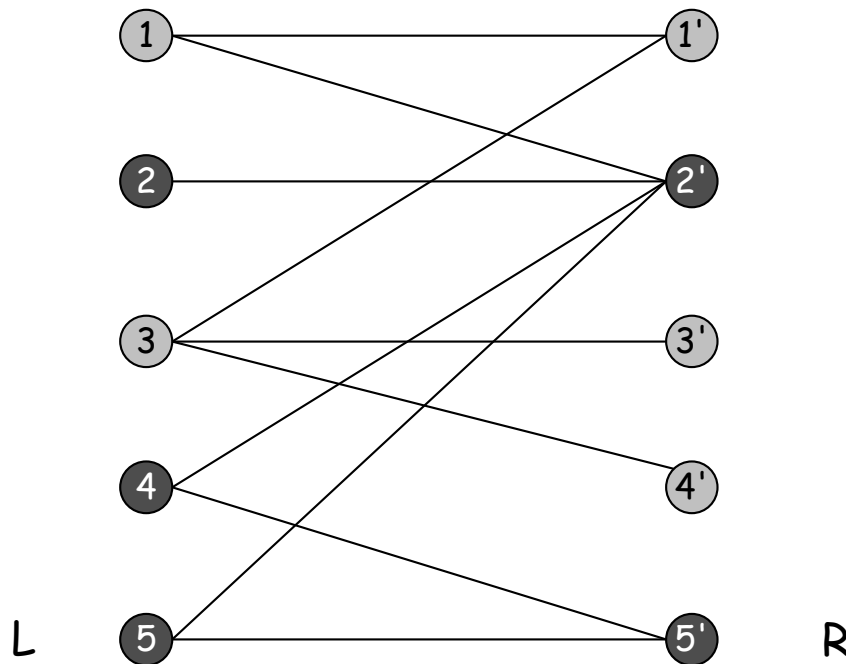
- Clearly we must have $|L| = |R|$.
- What other conditions are necessary?
- What conditions are sufficient?

Perfect Matching

Notation. Let S be a subset of nodes, and let $N(S)$ be the set of nodes adjacent to nodes in S .

Observation. If a bipartite graph $G = (L \cup R, E)$, has a perfect matching, then $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. Each node in S has to be matched to a different node in $N(S)$.



No perfect matching:

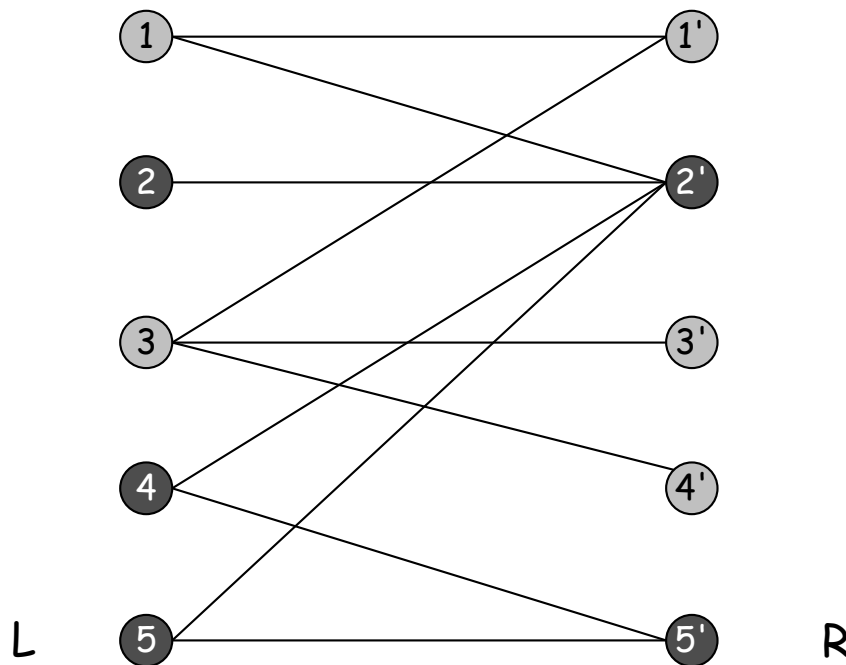
$S = \{ 2, 4, 5 \}$

$N(S) = \{ 2', 5' \}.$

Marriage Theorem

Marriage Theorem. [Frobenius 1917, Hall 1935] Let $G = (L \cup R, E)$ be a bipartite graph with $|L| = |R|$. Then, G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq L$.

Pf. \Rightarrow This was the previous observation.



No perfect matching:

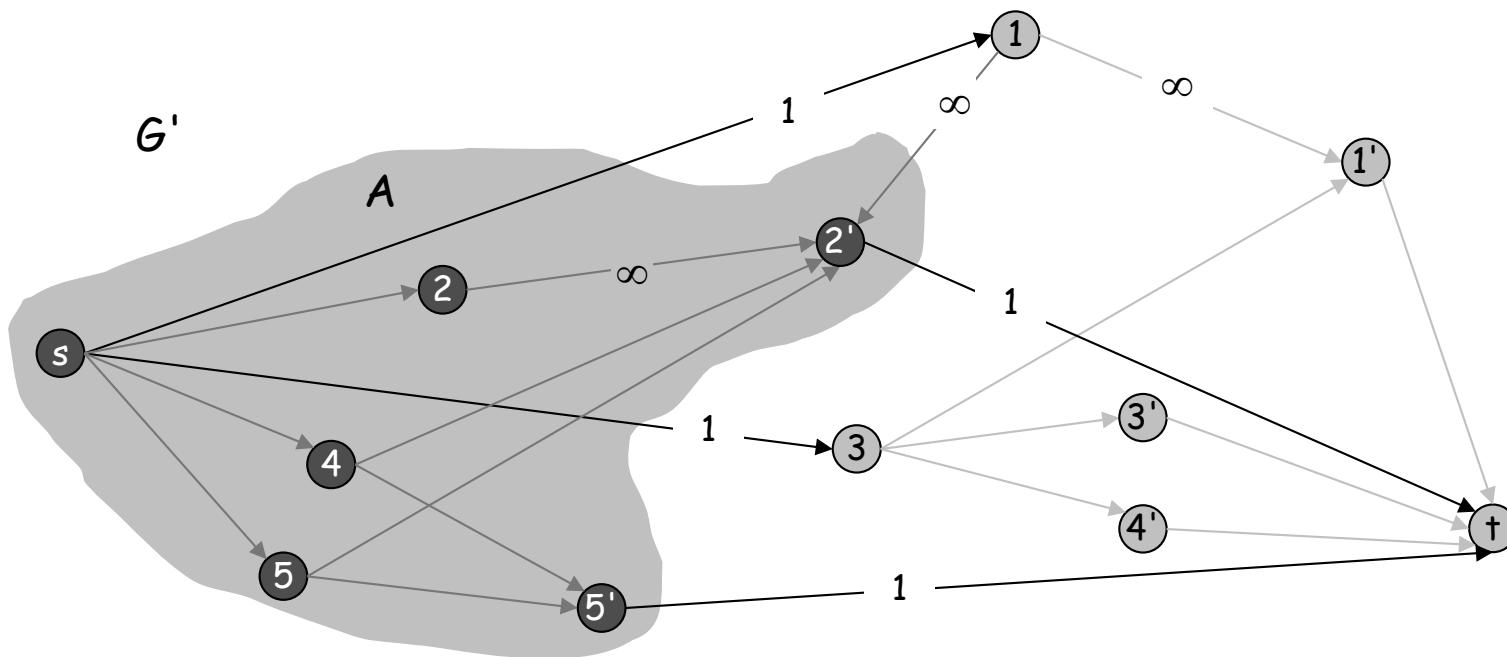
$S = \{ 2, 4, 5 \}$

$N(S) = \{ 2', 5' \}.$

Proof of Marriage Theorem

Pf. \Leftarrow Suppose G does not have a perfect matching.

- Formulate as a max flow problem and let (A, B) be min cut in G' .
- By max-flow min-cut, $\text{cap}(A, B) < |L|$.
- Define $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$.
- $\text{cap}(A, B) = |L_B| + |R_A|$.
- Since min cut can't use ∞ edges: $N(L_A) \subseteq R_A$.
- $|N(L_A)| \leq |R_A| = \text{cap}(A, B) - |L_B| < |L| - |L_B| = |L_A|$.
- Choose $S = L_A$. ▪



$L_A = \{2, 4, 5\}$
 $L_B = \{1, 3\}$
 $R_A = \{2', 5'\}$
 $N(L_A) = \{2', 5'\}$

Bipartite Matching: Running Time

Which max flow algorithm to use for bipartite matching?

- Generic augmenting path: $O(m \text{ val}(f^*)) = O(mn)$.
- Capacity scaling: $O(m^2 \log C) = O(m^2)$.
- Shortest augmenting path: $O(m n^{1/2})$.

Non-bipartite matching.

- Structure of non-bipartite graphs is more complicated, but well-understood. [Tutte-Berge, Edmonds-Galai]
- Blossom algorithm: $O(n^4)$. [Edmonds 1965]
- Best known: $O(m n^{1/2})$. [Micali-Vazirani 1980]

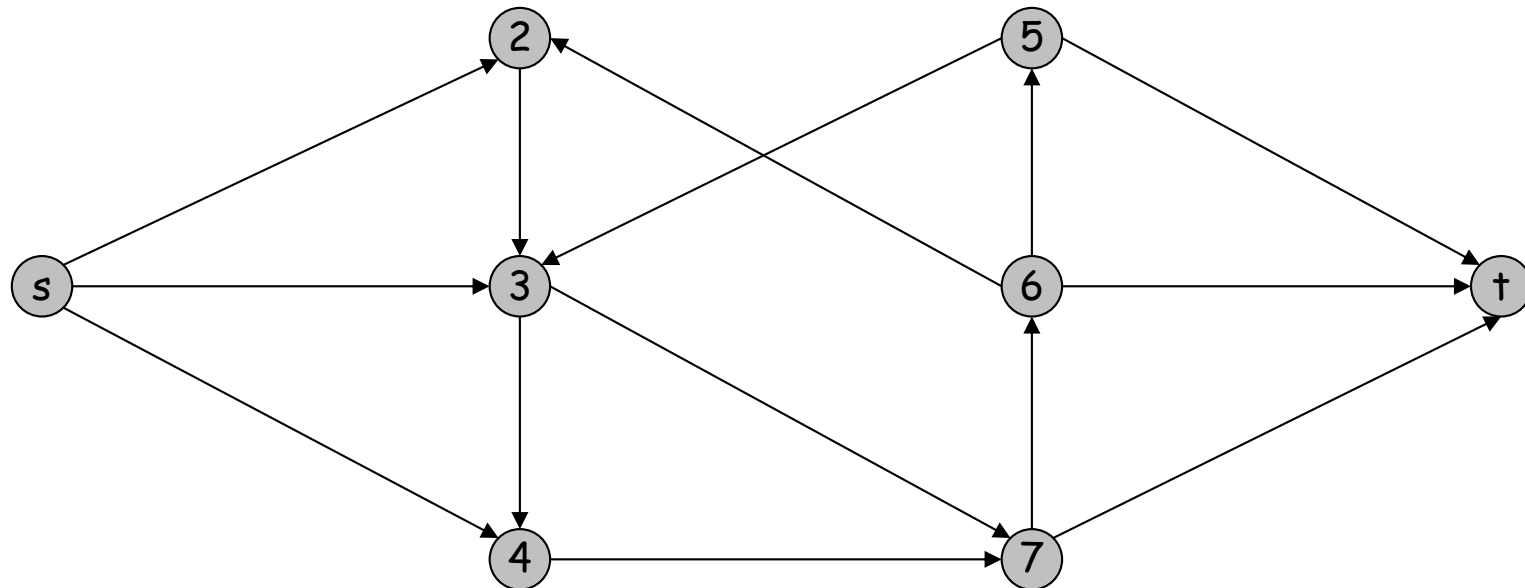
7.6 Disjoint Paths

Edge Disjoint Paths

Disjoint path problem. Given a digraph $G = (V, E)$ and two nodes s and t , find the max number of edge-disjoint s - t paths.

Def. Two paths are **edge-disjoint** if they have no edge in common.

Ex: communication networks.

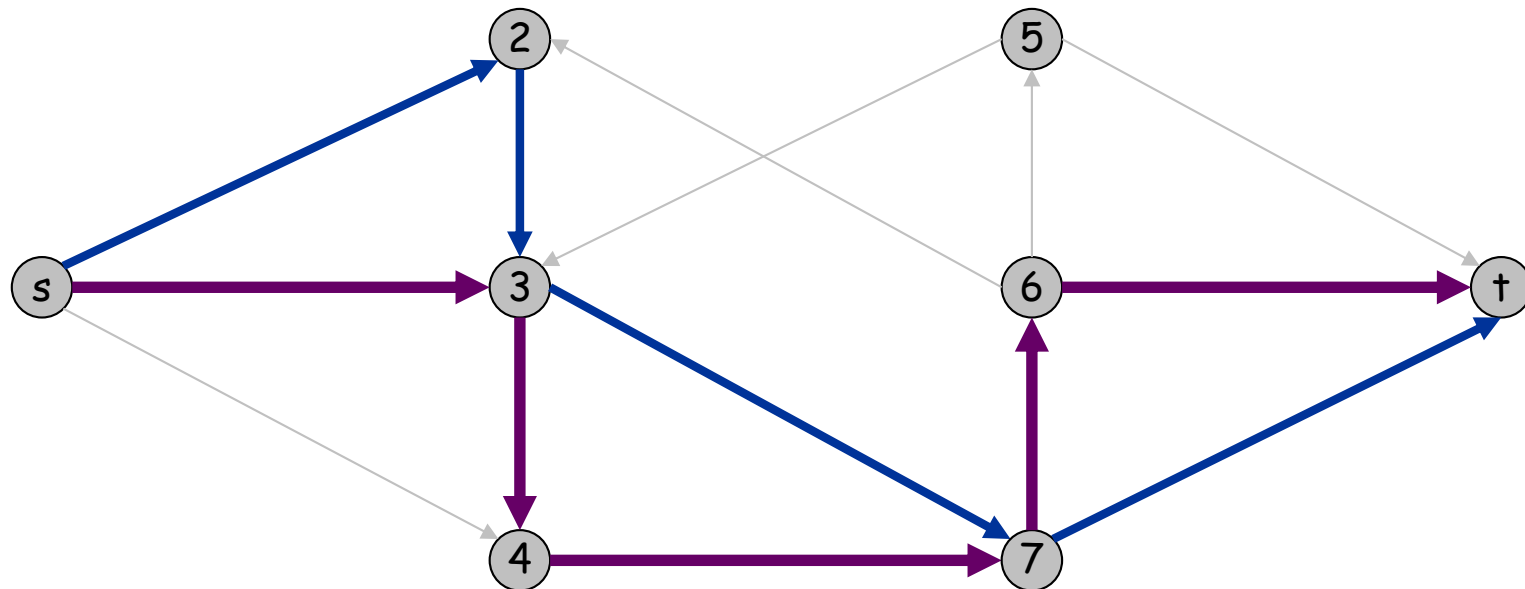


Edge Disjoint Paths

Disjoint path problem. Given a digraph $G = (V, E)$ and two nodes s and t , find the max number of edge-disjoint s - t paths.

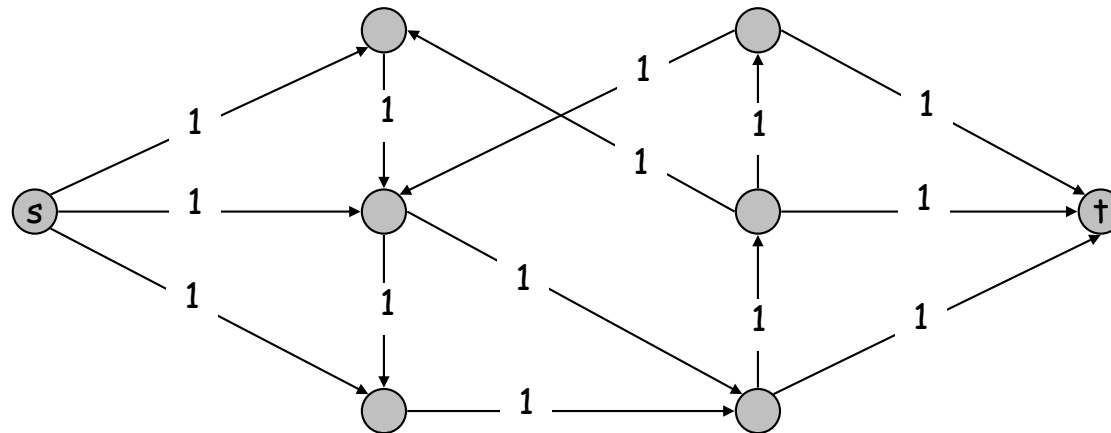
Def. Two paths are **edge-disjoint** if they have no edge in common.

Ex: communication networks.



Edge Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



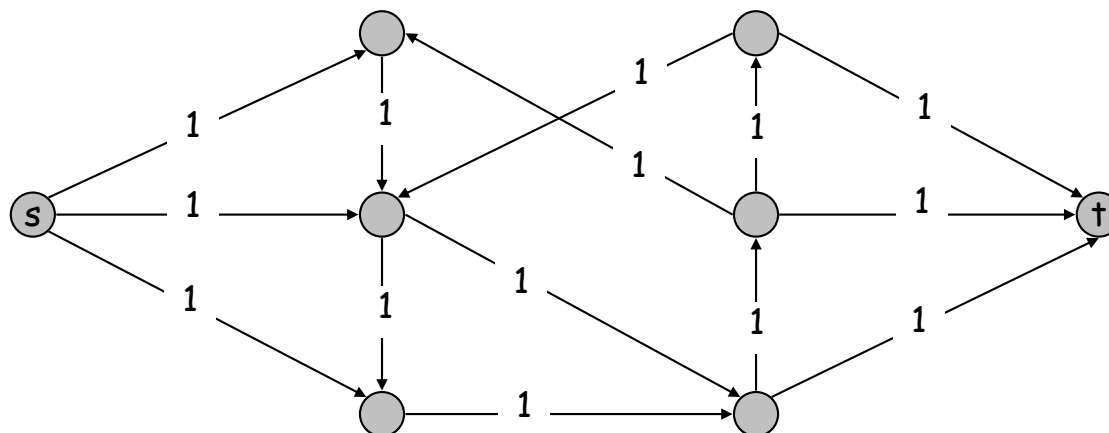
Theorem. Max number edge-disjoint s - t paths equals max flow value.

Pf. \leq

- Suppose there are k edge-disjoint paths P_1, \dots, P_k .
- Set $f(e) = 1$ if e participates in some path P_i ; else set $f(e) = 0$.
- Since paths are edge-disjoint, f is a flow of value k . ■

Edge Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



Theorem. Max number edge-disjoint s - t paths equals max flow value.

Pf. \geq

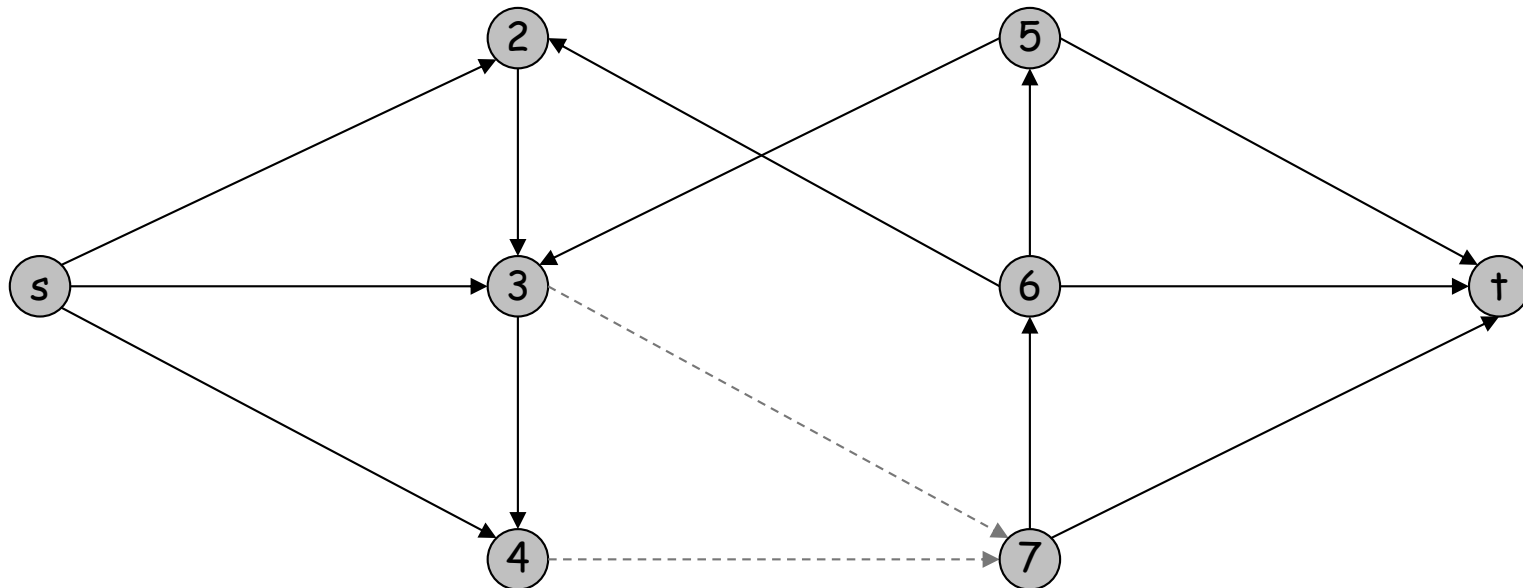
- Suppose max flow value is k .
- Integrality theorem \Rightarrow there exists 0-1 flow f of value k .
- Consider edge (s, u) with $f(s, u) = 1$.
 - by conservation, there exists an edge (u, v) with $f(u, v) = 1$
 - continue until reach t , always choosing a new edge
- Produces k (not necessarily simple) edge-disjoint paths. ■

can eliminate cycles to get simple paths if desired

Network Connectivity

Network connectivity. Given a digraph $G = (V, E)$ and two nodes s and t , find min number of edges whose removal disconnects t from s .

Def. A set of edges $F \subseteq E$ **disconnects t from s** if all s - t paths uses at least on edge in F .

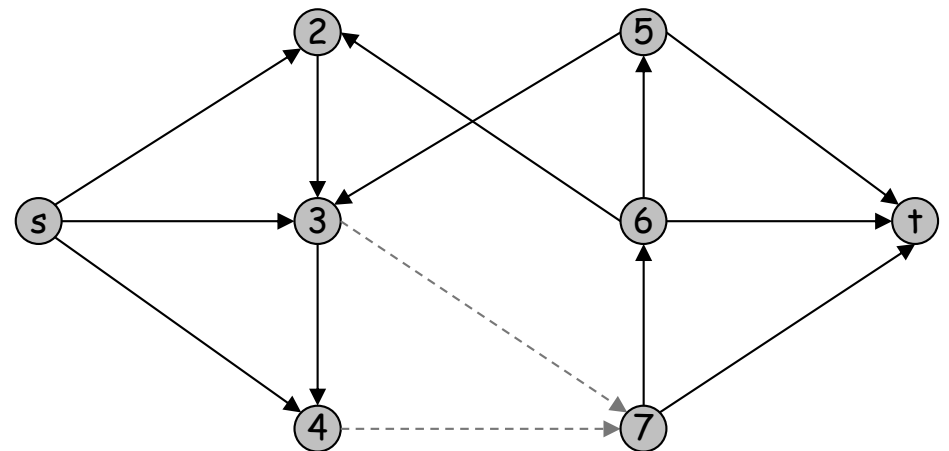
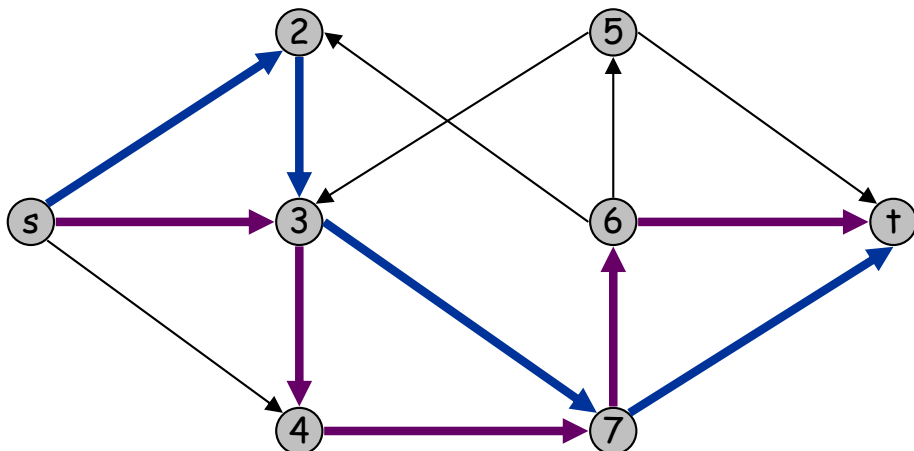


Edge Disjoint Paths and Network Connectivity

Theorem. [Menger 1927] The max number of edge-disjoint s - t paths is equal to the min number of edges whose removal disconnects t from s .

Pf. \leq

- Suppose the removal of $F \subseteq E$ disconnects t from s , and $|F| = k$.
- All s - t paths use at least one edge of F . Hence, the number of edge-disjoint paths is at most k . ■

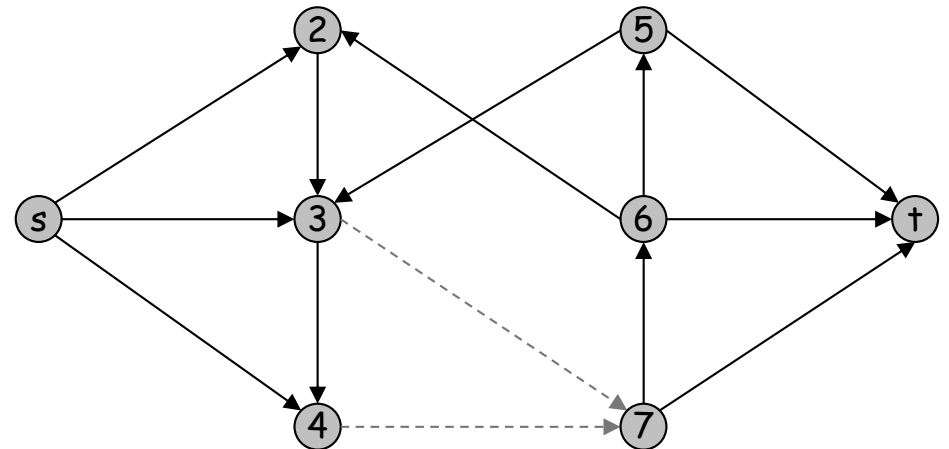
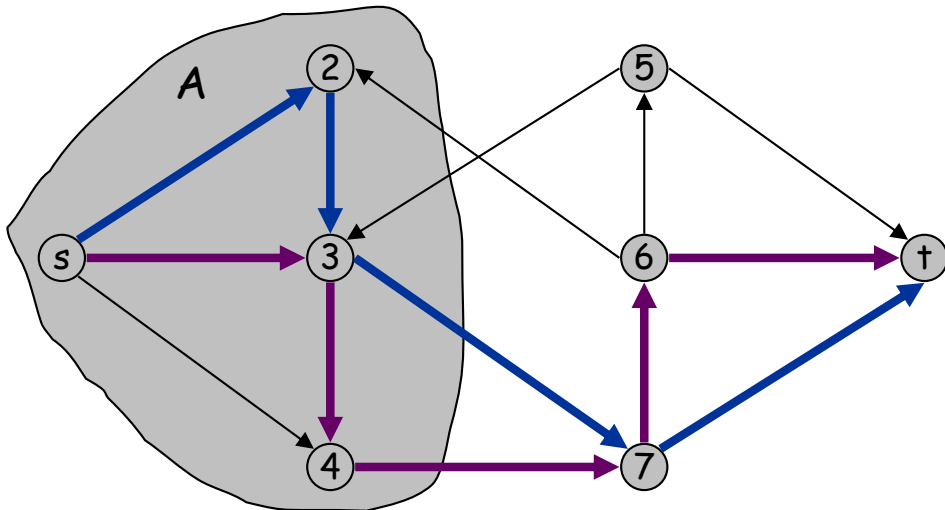


Disjoint Paths and Network Connectivity

Theorem. [Menger 1927] The max number of edge-disjoint s - t paths is equal to the min number of edges whose removal disconnects t from s .

Pf. \geq

- Suppose max number of edge-disjoint paths is k .
- Then max flow value is k .
- Max-flow min-cut \Rightarrow cut (A, B) of capacity k .
- Let F be set of edges going from A to B .
- $|F| = k$ and disconnects t from s . ■



7.7 Extensions to Max Flow

Circulation with Demands

Circulation with demands.

- Directed graph $G = (V, E)$.
- Edge capacities $c(e)$, $e \in E$.
- Node supply and demands $d(v)$, $v \in V$.

↑
demand if $d(v) > 0$; supply if $d(v) < 0$; transshipment if $d(v) = 0$

Def. A **circulation** is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

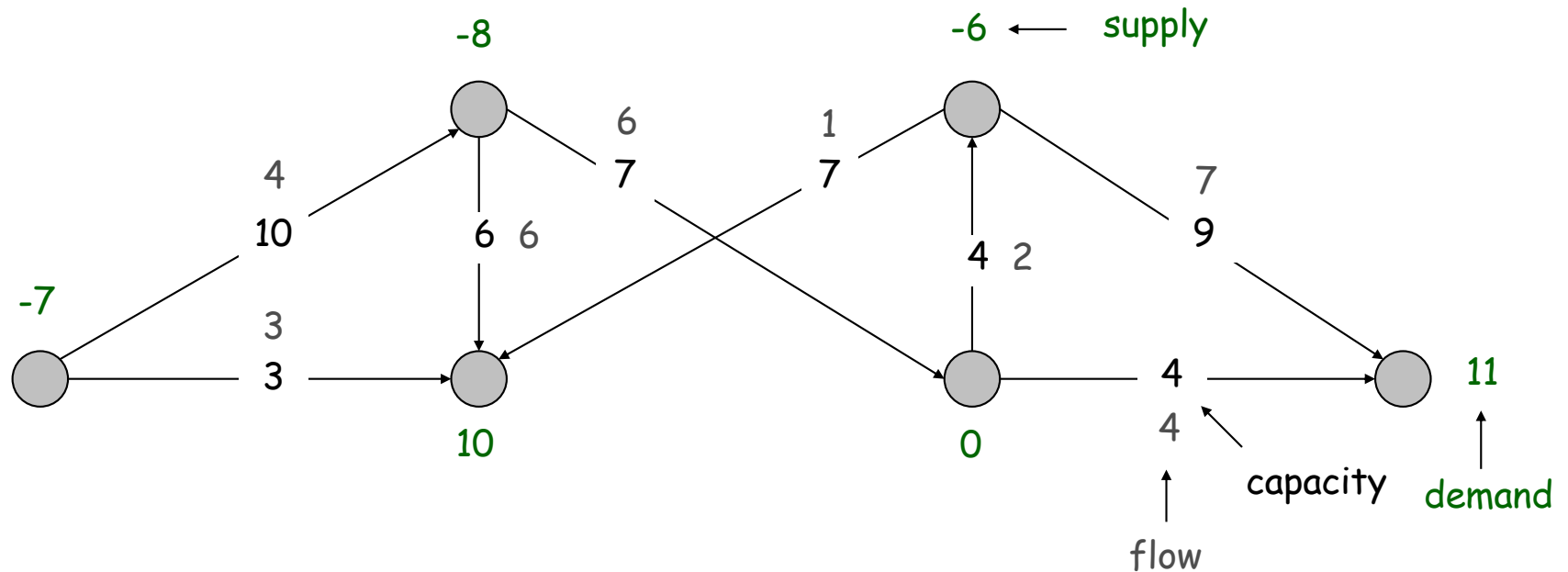
Circulation problem: given (V, E, c, d) , does there exist a circulation?

Circulation with Demands

Necessary condition: sum of supplies = sum of demands.

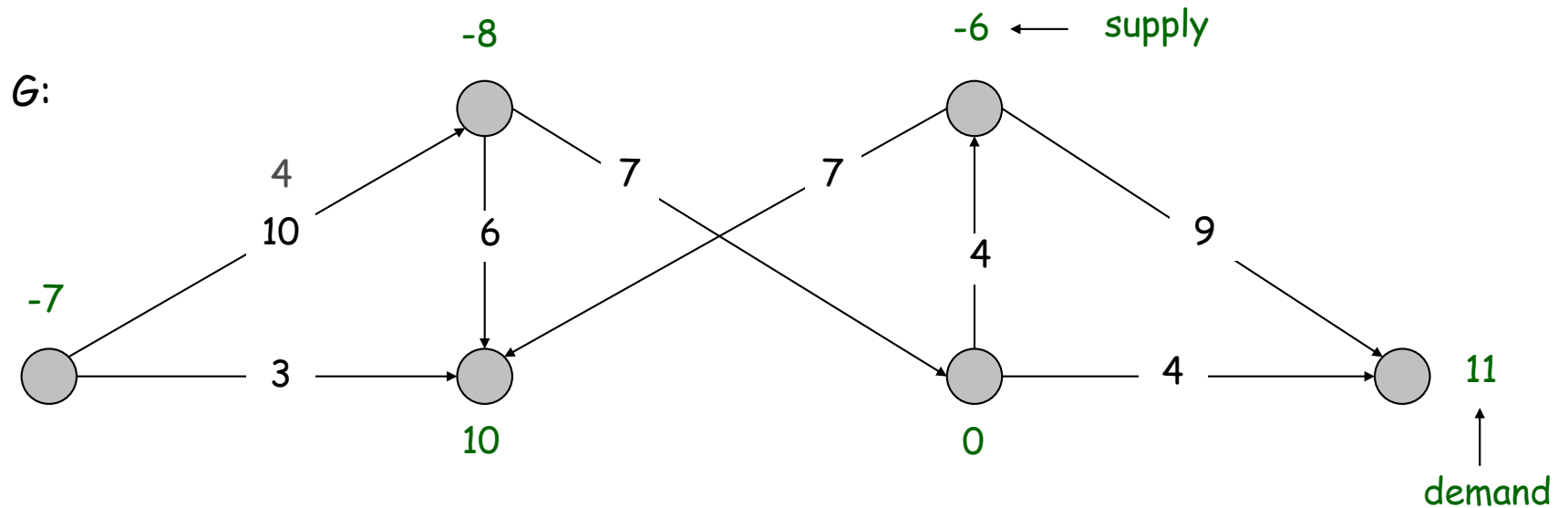
$$\sum_{v: d(v) > 0} d(v) = \sum_{v: d(v) < 0} -d(v) =: D$$

Pf. Sum conservation constraints for every demand node v .



Circulation with Demands

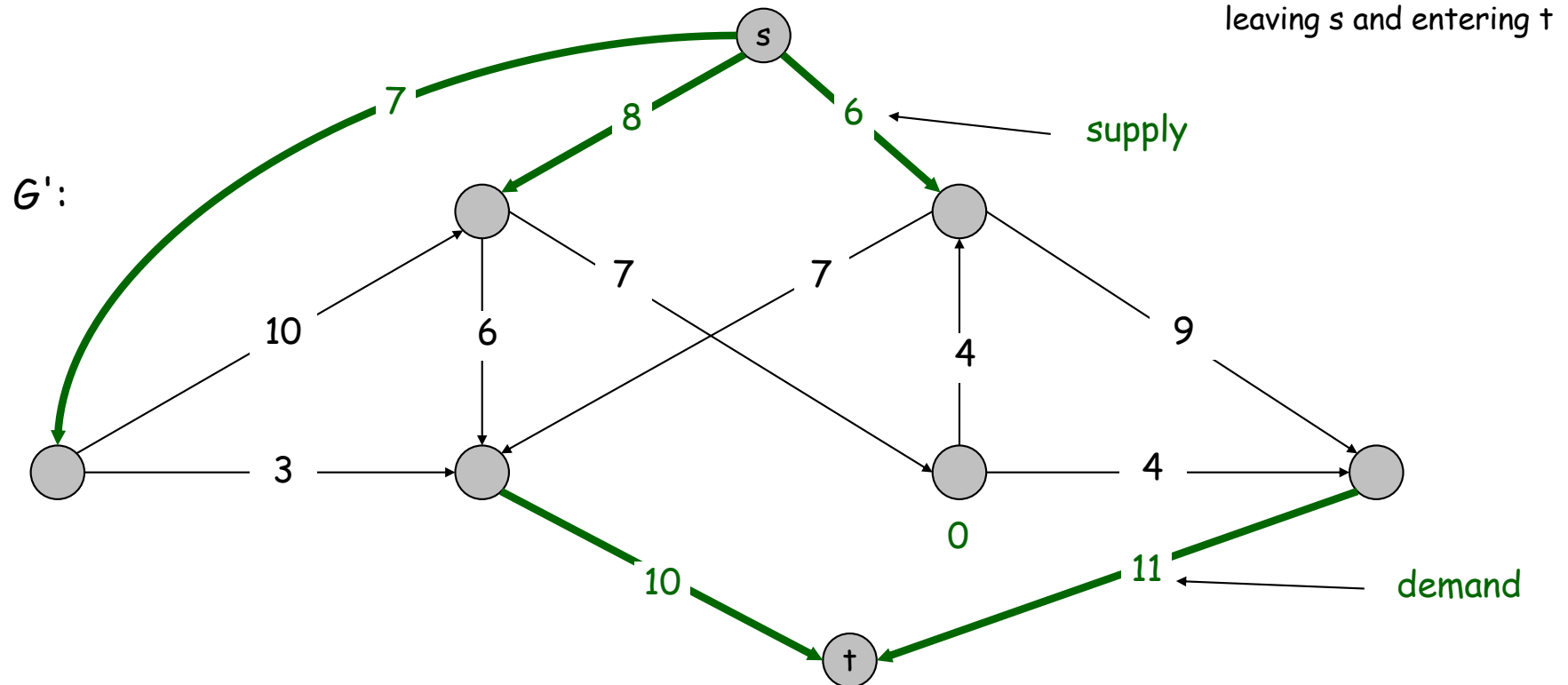
Max flow formulation.



Circulation with Demands

Max flow formulation.

- Add new source s and sink t .
- For each v with $d(v) < 0$, add edge (s, v) with capacity $-d(v)$.
- For each v with $d(v) > 0$, add edge (v, t) with capacity $d(v)$.
- Claim: G has circulation iff G' has max flow of value D .



Circulation with Demands

Integrality theorem. If all capacities and demands are integers, and there exists a circulation, then there exists one that is integer-valued.

Pf. Follows from max flow formulation and integrality theorem for max flow.

Characterization. Given (V, E, c, d) , there does **not** exist a circulation iff there exists a node partition (A, B) such that $\sum_{v \in B} d_v > \text{cap}(A, B)$

Pf idea. Look at min cut in G' .

↑
demand by nodes in B exceeds supply
of nodes in B plus max capacity of
edges going from A to B

Circulation with Demands and Lower Bounds

Feasible circulation.

- Directed graph $G = (V, E)$.
- Edge capacities $c(e)$ and lower bounds $\ell(e)$, $e \in E$.
- Node supply and demands $d(v)$, $v \in V$.

Def. A **circulation** is a function that satisfies:

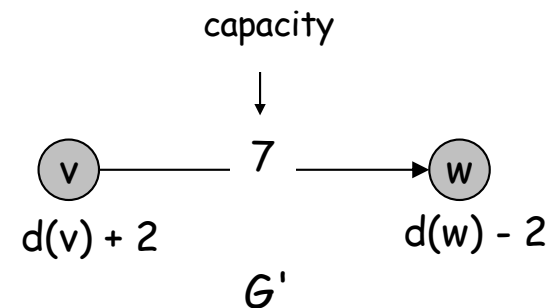
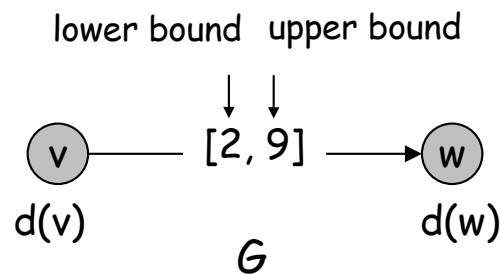
- For each $e \in E$: $\ell(e) \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (conservation)

Circulation problem with lower bounds. Given (V, E, ℓ, c, d) , does there exists a a circulation?

Circulation with Demands and Lower Bounds

Idea. Model lower bounds with demands.

- Send $\ell(e)$ units of flow along edge e .
- Update demands of both endpoints.



Theorem. There exists a circulation in G iff there exists a circulation in G' . If all demands, capacities, and lower bounds in G are integers, then there is a circulation in G that is integer-valued.

Pf sketch. $f(e)$ is a circulation in G iff $f'(e) = f(e) - \ell(e)$ is a circulation in G' .

7.8 Survey Design

Survey Design

Survey design.

- Design survey asking n_1 consumers about n_2 products.
- Can only survey consumer i about a product j if they own it.
- Ask consumer i between c_i and c_i' questions.
- Ask between p_j and p_j' consumers about product j .

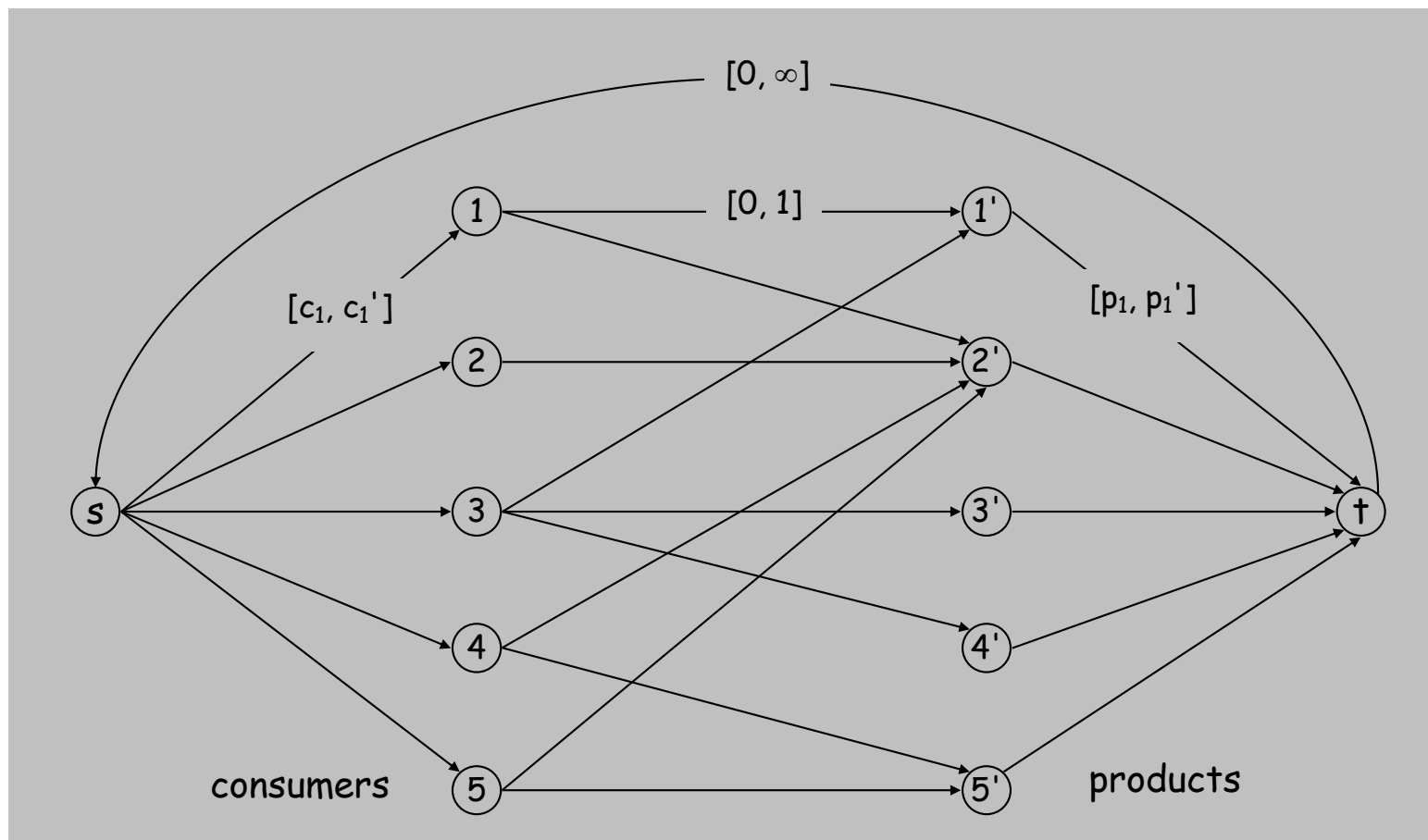
Goal. Design a survey that meets these specs, if possible.

Bipartite perfect matching. Special case when $c_i = c_i' = p_i = p_i' = 1$.

Survey Design

Algorithm. Formulate as a circulation problem with lower bounds.

- Include an edge (i, j) if customer own product i .
- Integer circulation \Leftrightarrow feasible survey design.



7.10 Image Segmentation

Image Segmentation

Image segmentation.

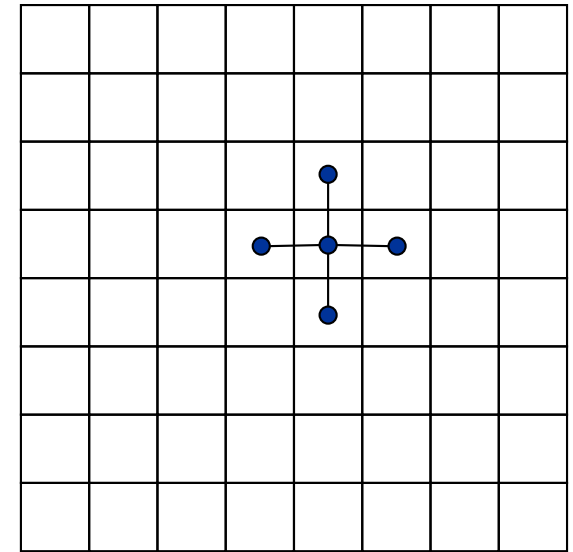
- Central problem in image processing.
- Divide image into coherent regions.

Ex: Three people standing in front of complex background scene.
Identify each person as a coherent object.

Image Segmentation

Foreground / background segmentation.

- Label each pixel in picture as belonging to foreground or background.
- V = set of pixels, E = pairs of neighboring pixels.
- $a_i \geq 0$ is likelihood pixel i in foreground.
- $b_i \geq 0$ is likelihood pixel i in background.
- $p_{ij} \geq 0$ is separation penalty for labeling one of i and j as foreground, and the other as background.



Goals.

- Accuracy: if $a_i > b_i$ in isolation, prefer to label i in foreground.
- Smoothness: if many neighbors of i are labeled foreground, we should be inclined to label i as foreground.

- Find partition (A, B) that maximizes:

$$\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

\nearrow
foreground

\nwarrow
background

Image Segmentation

Formulate as min cut problem.

- Maximization.
- No source or sink.
- Undirected graph.

Turn into minimization problem.

- Maximizing
$$\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

is equivalent to minimizing
$$\underbrace{\left(\sum_{i \in V} a_i + \sum_{j \in V} b_j \right)}_{\text{a constant}} - \sum_{i \in A} a_i - \sum_{j \in B} b_j + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

- or alternatively
$$\sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

Image Segmentation

Formulate as min cut problem.

- $G' = (V', E')$.
- Add source to correspond to foreground;
add sink to correspond to background
- Use two anti-parallel edges instead of undirected edge.

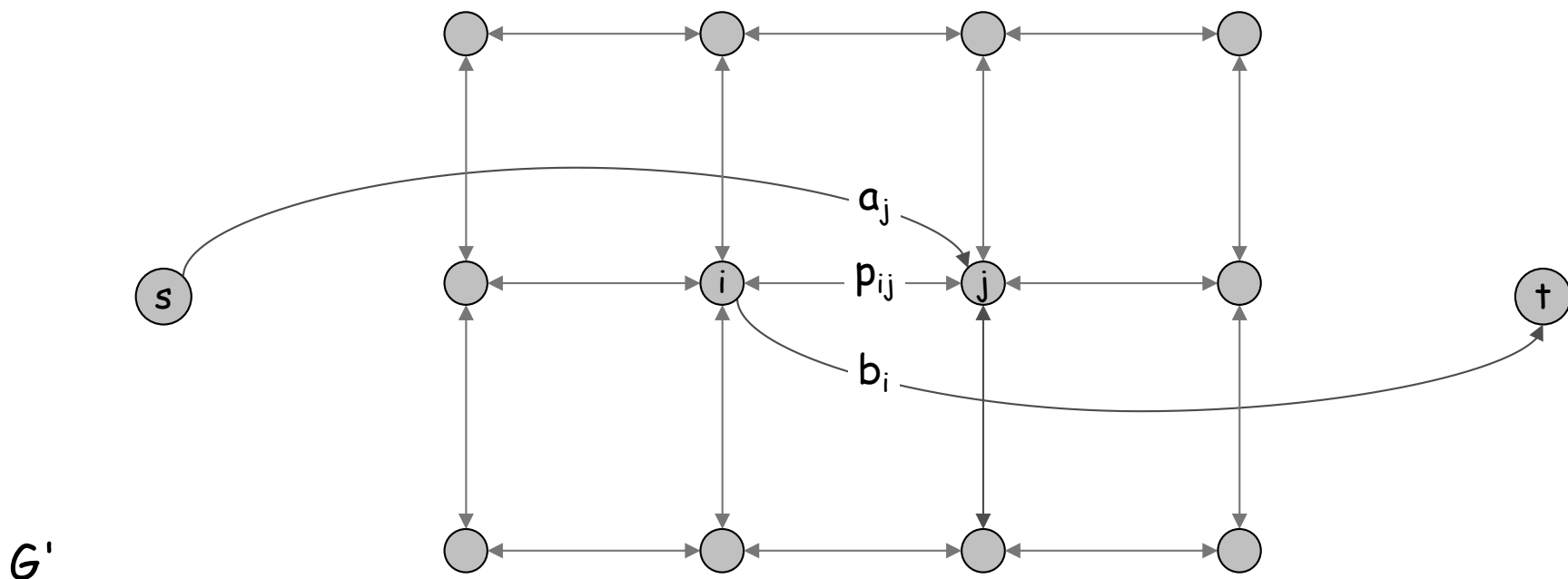
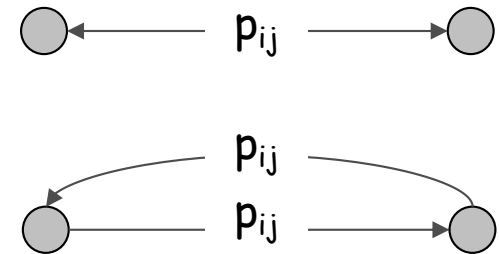


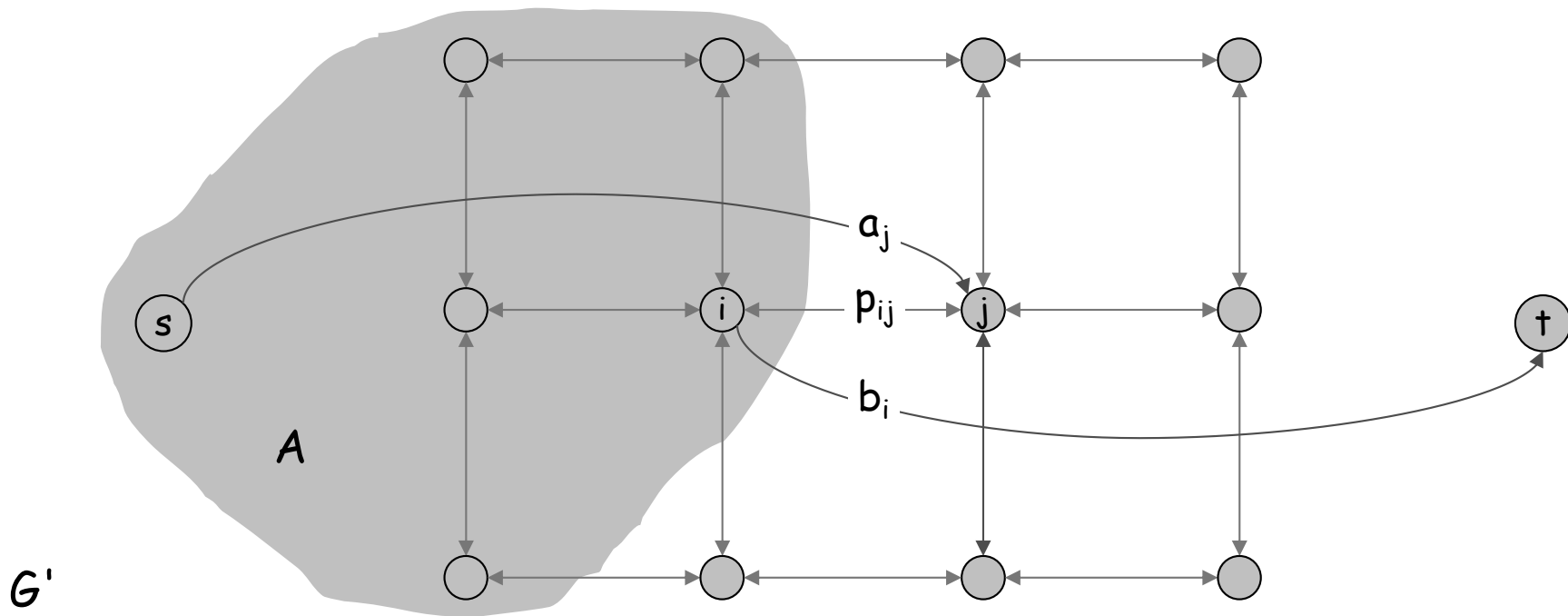
Image Segmentation

Consider min cut (A, B) in G' .

- A = foreground.

$$cap(A, B) = \sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E \\ i \in A, j \in B}} p_{ij} \quad \leftarrow \begin{array}{l} \text{if } i \text{ and } j \text{ on different sides,} \\ p_{ij} \text{ counted exactly once} \end{array}$$

- Precisely the quantity we want to minimize.



7.11 Project Selection

Project Selection

Projects with prerequisites.

can be positive or negative



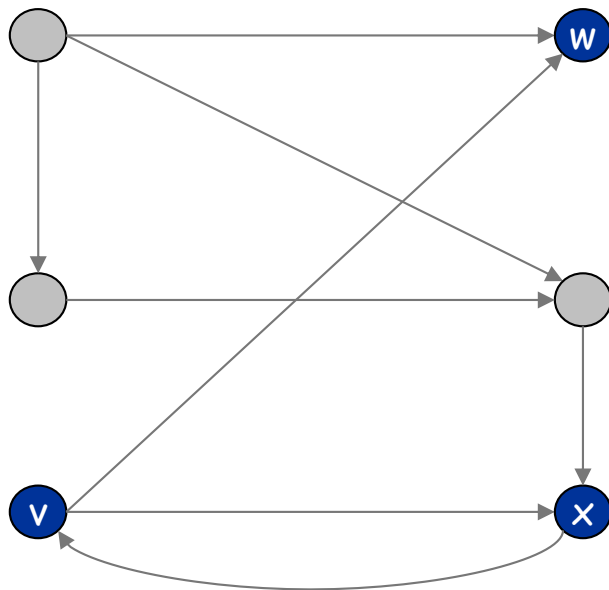
- Set P of possible projects. Project v has associated revenue p_v .
 - some projects generate money: create interactive e-commerce interface, redesign web page
 - others cost money: upgrade computers, get site license
- Set of prerequisites E . If $(v, w) \in E$, can't do project v and unless also do project w .
- A subset of projects $A \subseteq P$ is **feasible** if the prerequisite of every project in A also belongs to A .

Project selection. Choose a feasible subset of projects to maximize revenue.

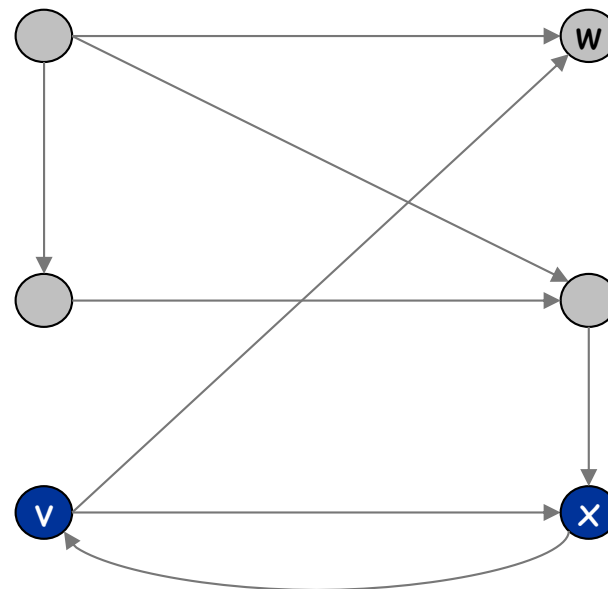
Project Selection: Prerequisite Graph

Prerequisite graph.

- Include an edge from v to w if can't do v without also doing w .
- $\{v, w, x\}$ is feasible subset of projects.
- $\{v, x\}$ is infeasible subset of projects.



feasible

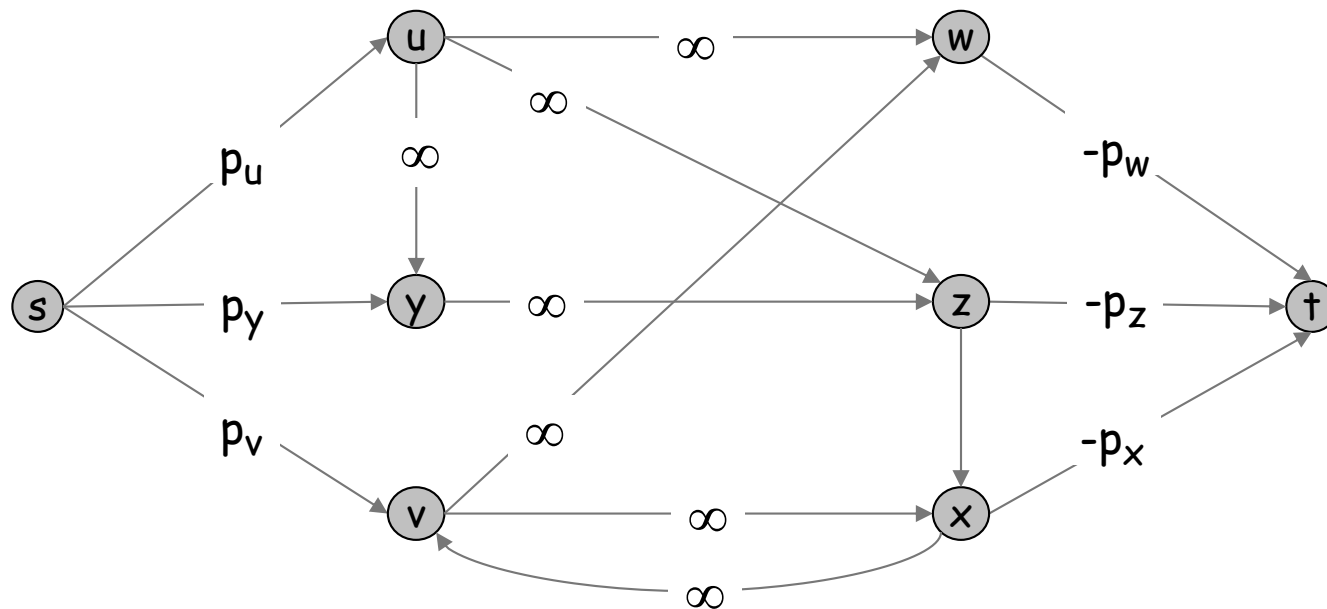


infeasible

Project Selection: Min Cut Formulation

Min cut formulation.

- Assign capacity ∞ to all prerequisite edge.
- Add edge (s, v) with capacity p_v if $p_v > 0$.
- Add edge (v, t) with capacity $-p_v$ if $p_v < 0$.
- For notational convenience, define $p_s = p_t = 0$.

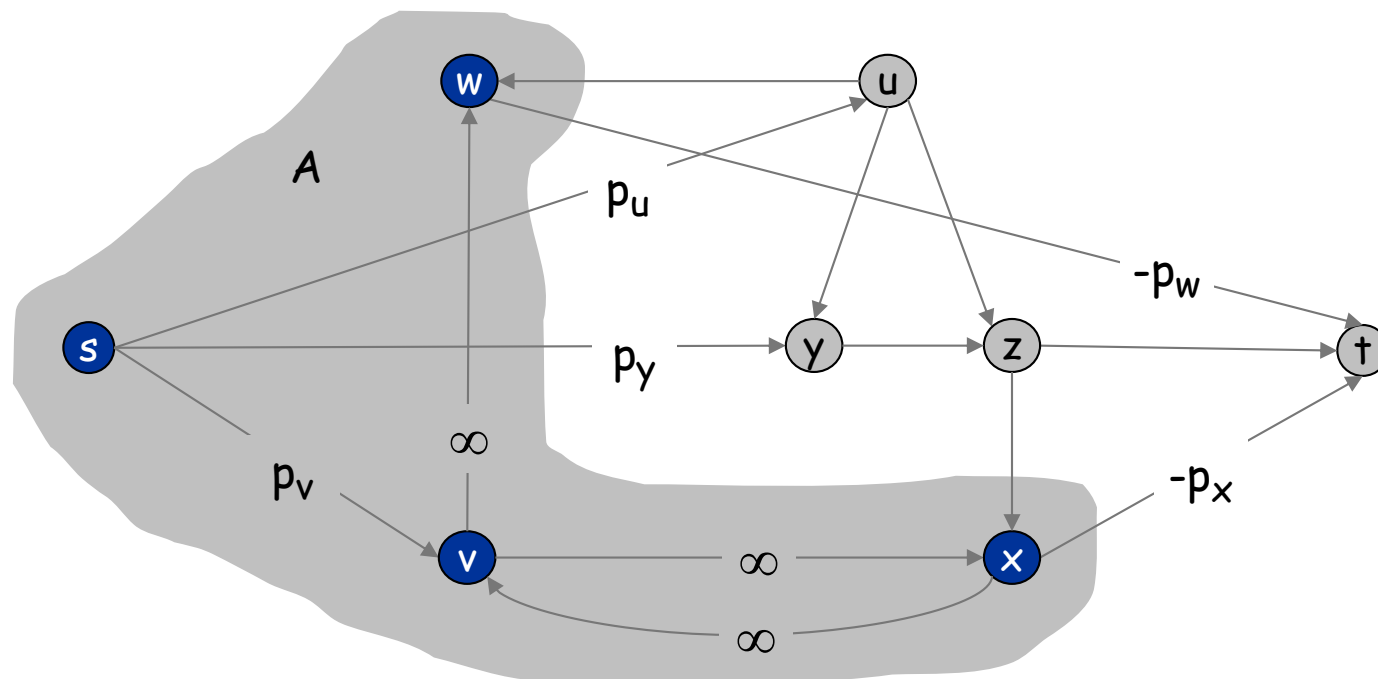


Project Selection: Min Cut Formulation

Claim. (A, B) is min cut iff $A - \{s\}$ is optimal set of projects.

- Infinite capacity edges ensure $A - \{s\}$ is feasible.
- Max revenue because:

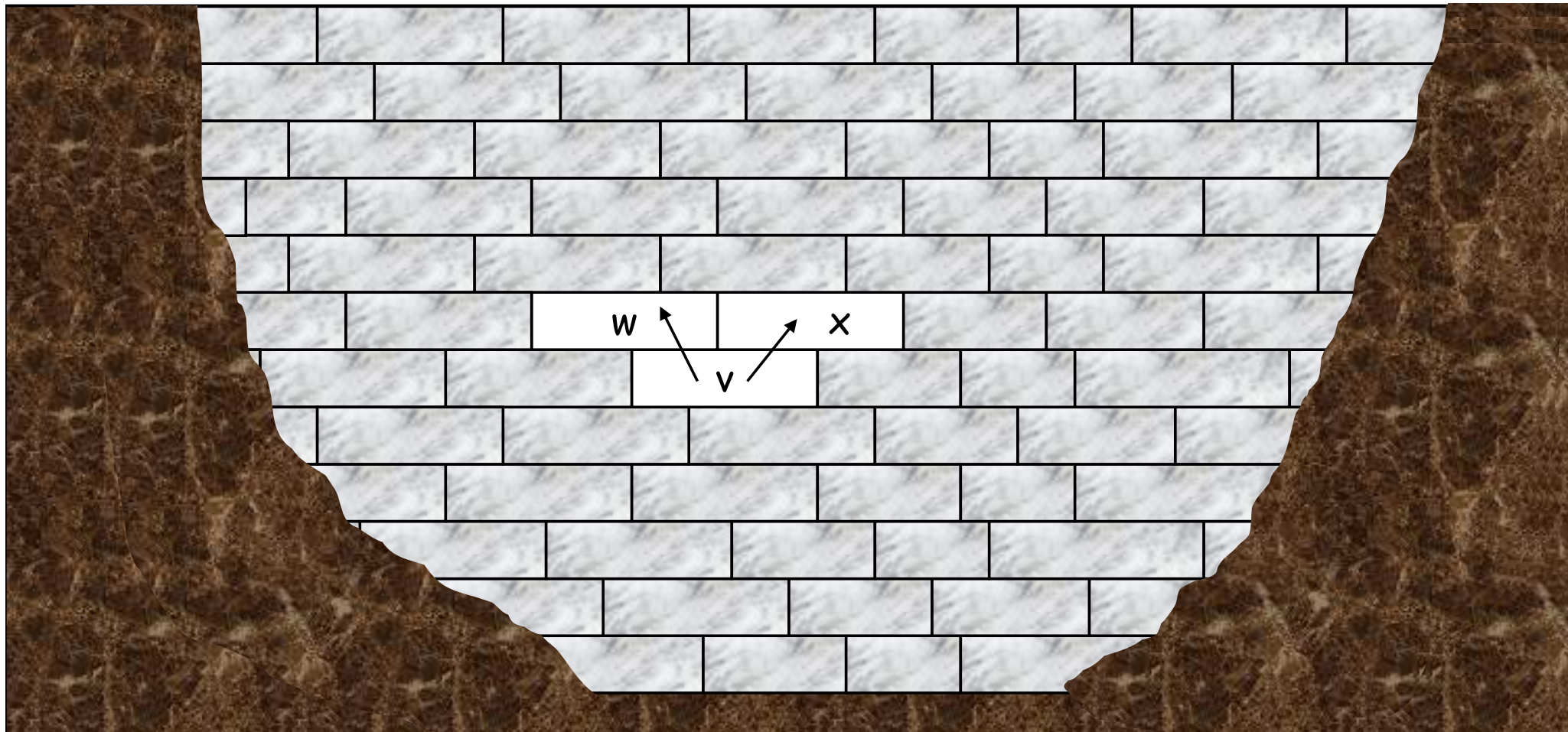
$$\begin{aligned} \text{cap}(A, B) &= \sum_{v \in B: p_v > 0} p_v + \sum_{v \in A: p_v < 0} (-p_v) \\ &= \underbrace{\sum_{v: p_v > 0} p_v}_{\text{constant}} - \sum_{v \in A} p_v \end{aligned}$$



Open Pit Mining

Open-pit mining. (studied since early 1960s)

- Blocks of earth are extracted from surface to retrieve ore.
- Each block v has net value $p_v = \text{value of ore} - \text{processing cost}$.
- Can't remove block v before w or x .



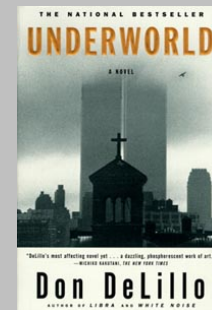
7.12 Baseball Elimination

"See that thing in the paper last week about Einstein? . . . Some reporter asked him to figure out the mathematics of the pennant race. You know, one team wins so many of their remaining games, the other teams win this number or that number. What are the myriad possibilities? Who's got the edge?"

"The hell does he know?"

"Apparently not much. He picked the Dodgers to eliminate the Giants last Friday."

- Don DeLillo, *Underworld*



Baseball Elimination

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}			
				Atl	Phi	NY	Mon
Atlanta	83	71	8	-	1	6	1
Philly	80	79	3	1	-	0	2
New York	78	78	6	6	0	-	0
Montreal	77	82	3	1	2	0	-

Which teams have a chance of finishing the season with most wins?

- Montreal eliminated since it can finish with at most 80 wins, but Atlanta already has 83.
- $w_i + r_i < w_j \Rightarrow$ team i eliminated.
- Only reason sports writers appear to be aware of.
- Sufficient, but not necessary!

Baseball Elimination

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}			
				Atl	Phi	NY	Mon
Atlanta	83	71	8	-	1	6	1
Philly	80	79	3	1	-	0	2
New York	78	78	6	6	0	-	0
Montreal	77	82	3	1	2	0	-

Which teams have a chance of finishing the season with most wins?

- Philly can win 83, but still eliminated . . .
- If Atlanta loses a game, then some other team wins one.

Remark. Answer depends not just on **how many** games already won and left to play, but also on **whom** they're against.

Baseball Elimination

TUESDAY, SEPTEMBER 10, 1996

San Francisco Chronicle

The Gate

Sports Online

► <http://www.sfgate.com>

SPORTING G

49ers, Young Get Big Breac



Quarterback m

By Gary Swan
Chronicle Staff Writer

The bye week has come at a perfect time for the 49ers and quarterback Steve Young. If they had a game next Sunday, there's a good chance Young would not play.

But the pulled groin muscle on his up-

Giants Officially Leave the NL West Race

By Nancy Gay
Chronicle Staff Writer

With the smack of another National League West bat 500 miles away, the Giants' run at the division title ended last night, just as they were handing the visiting St. Louis Cardinals an even bigger lead in the NL Central.

CARDINALS 6
GIANTS 2

In San Diego, Greg Vaughn's three-run homer in the eighth pushed the Padres over the Pirates and officially shoved the rest of the Giants' season into the back-ground. On the heels of their tedious 6-2 loss before an announced crowd of 10,307 at Candlestick Park, the Giants fell 19½ games off the lead.

As it is, the worst the Padres (80-65) can finish is 80-82. The Giants have fallen to 59-83 with 20

Financing in Place
For Giants' New Stadium
SEE PAGE B1, MAIN NEWS

games left; they cannot win 80 games. Coming off a miserable 2-8 mark on a three-city road trip that saw their road record drop to 27-47, the Giants were hoping to get off on the right foot in their longest homestand of the year (15 games, 14 days).

"Where we are, you're going to be eliminated sooner or later," Baker said quietly. "But it doesn't alter the fact that we've still got to play ball. You've still got to play hard, the fans come out to watch you play. You've got to play for the fact of loving to play, no matter where you are in the standings.

"You've got to play the role of spoiler, to not make it easier on
GIANTS: Page D5 Col. 3

Baseball Elimination

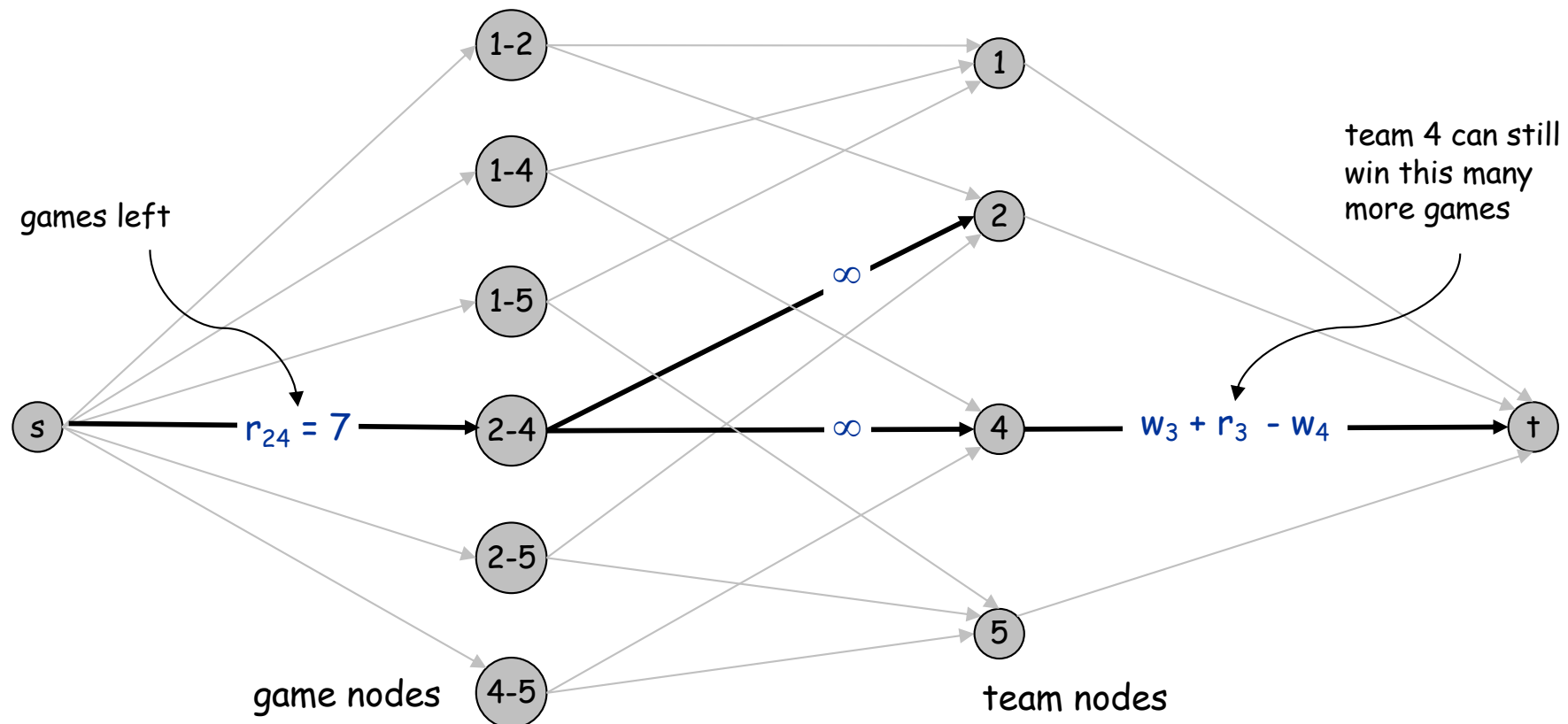
Baseball elimination problem.

- Set of teams S .
- Distinguished team $s \in S$.
- Team x has won w_x games already.
- Teams x and y play each other r_{xy} additional times.
- Is there any outcome of the remaining games in which team s finishes with the most (or tied for the most) wins?

Baseball Elimination: Max Flow Formulation

Can team 3 finish with most wins?

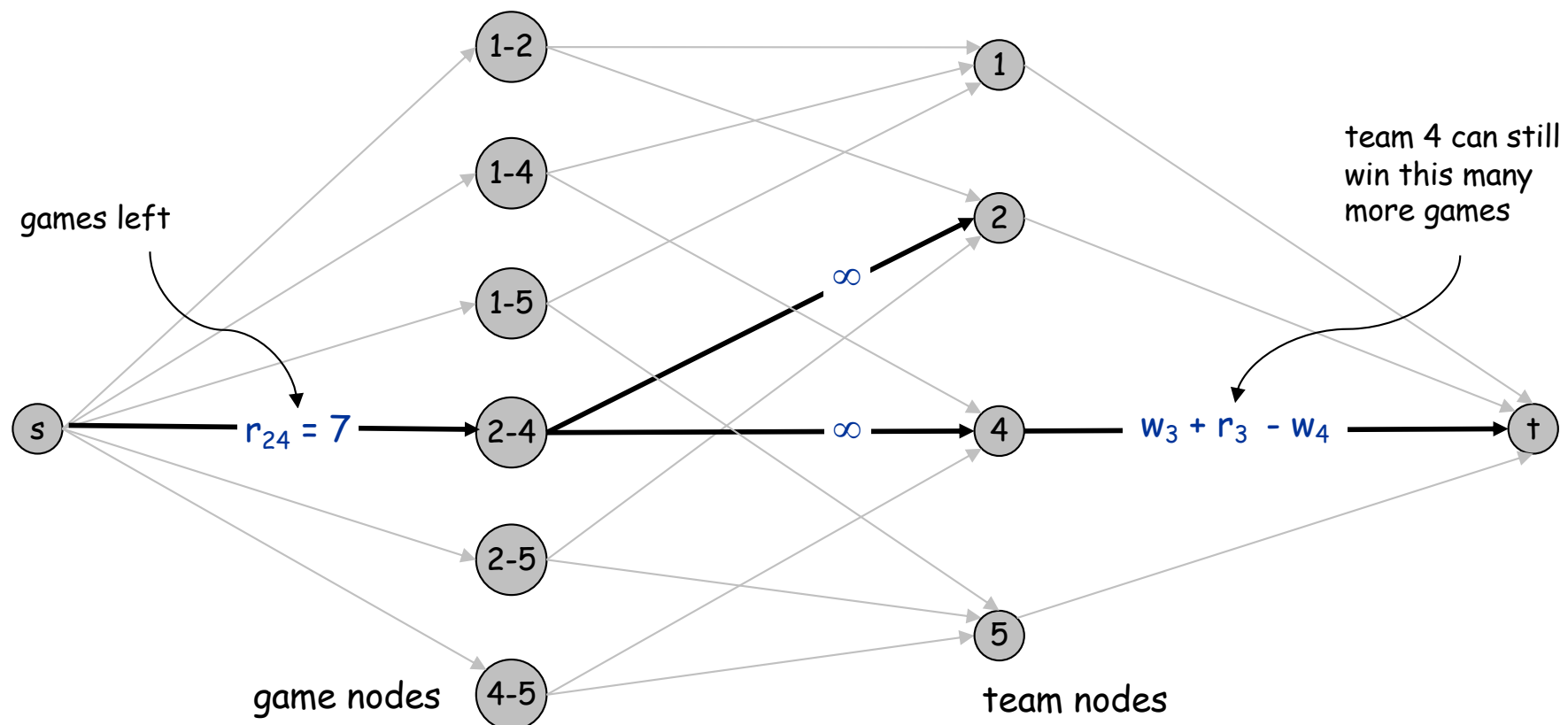
- Assume team 3 wins all remaining games $\Rightarrow w_3 + r_3$ wins.
- Divvy remaining games so that all teams have $\leq w_3 + r_3$ wins.



Baseball Elimination: Max Flow Formulation

Theorem. Team 3 is not eliminated iff max flow saturates all edges leaving source.

- Integrality theorem \Rightarrow each remaining game between x and y added to number of wins for team x or team y .
- Capacity on (x, t) edges ensure no team wins too many games.



Baseball Elimination: Explanation for Sports Writers

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}				
				NY	Bal	Bos	Tor	Det
NY	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	-	-
Detroit	49	86	27	3	4	0	0	-

AL East: August 30, 1996

Which teams have a chance of finishing the season with most wins?

- Detroit could finish season with $49 + 27 = 76$ wins.

Baseball Elimination: Explanation for Sports Writers

Team i	Wins w_i	Losses l_i	To play r_i	Against = r_{ij}				
				NY	Bal	Bos	Tor	Det
NY	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	-	-
Detroit	49	86	27	3	4	0	0	-

AL East: August 30, 1996

Which teams have a chance of finishing the season with most wins?

- Detroit could finish season with $49 + 27 = 76$ wins.

Certificate of elimination. $R = \{\text{NY, Bal, Bos, Tor}\}$

- Have already won $w(R) = 278$ games.
- Must win at least $r(R) = 27$ more.
- Average team in R wins at least $305/4 > 76$ games.

Baseball Elimination: Explanation for Sports Writers

Certificate of elimination.

$$T \subseteq S, \quad w(T) := \overbrace{\sum_{i \in T} w_i}^{\# \text{ wins}}, \quad g(T) := \overbrace{\sum_{\{x,y\} \subseteq T} g_{xy}}^{\# \text{ remaining games}},$$

If $\overbrace{\frac{w(T) + g(T)}{|T|}}^{\text{LB on avg \# games won}} > w_z + g_z$ then z is **eliminated** (by subset T).

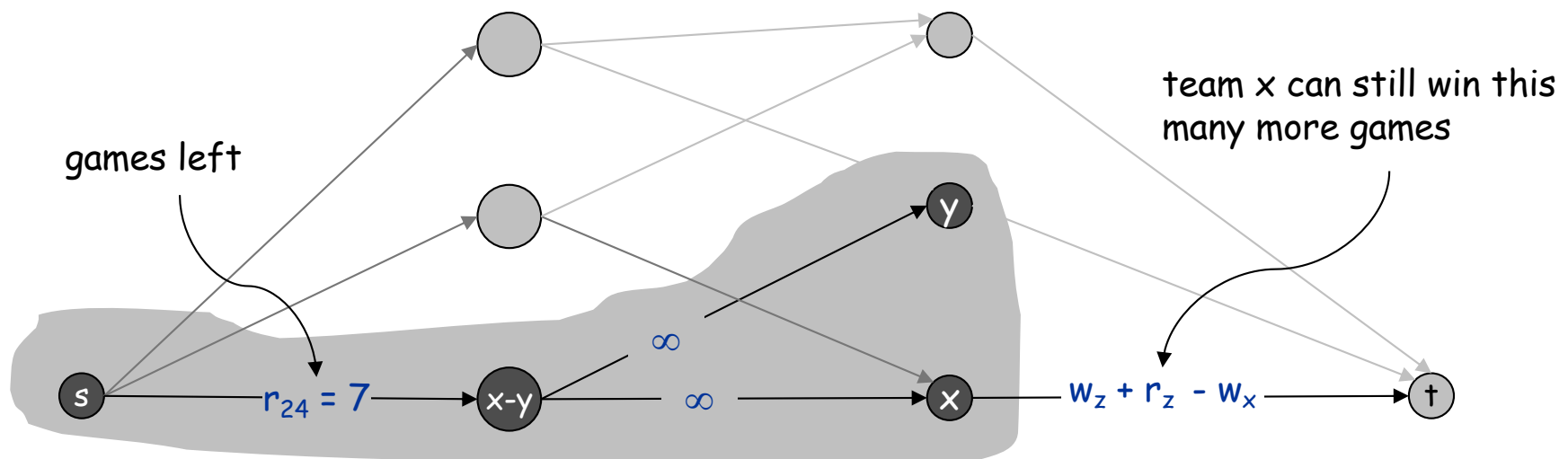
Theorem. [Hoffman-Rivlin 1967] Team z is eliminated iff there exists a subset T^* that eliminates z .

Proof idea. Let T^* = team nodes on source side of min cut.

Baseball Elimination: Explanation for Sports Writers

Pf of theorem.

- Use max flow formulation, and consider min cut (A, B) .
- Define T^* = team nodes on source side of min cut.
- Observe $x-y \in A$ iff both $x \in T^*$ and $y \in T^*$.
 - infinite capacity edges ensure if $x-y \in A$ then $x \in A$ and $y \in A$
 - if $x \in A$ and $y \in A$ but $x-y \in T$, then adding $x-y$ to A decreases capacity of cut



Baseball Elimination: Explanation for Sports Writers

Pf of theorem.

- Use max flow formulation, and consider min cut (A, B) .
- Define T^* = team nodes on source side of min cut.
- Observe $x-y \in A$ iff both $x \in T^*$ and $y \in T^*$.
- $g(S - \{z\}) > \text{cap}(A, B)$

$$\begin{aligned}
 &= \overbrace{g(S - \{z\}) - g(T^*)}^{\text{capacity of game edges leaving } s} + \overbrace{\sum_{x \in T^*} (w_z + g_z - w_x)}^{\text{capacity of team edges leaving } s} \\
 &= g(S - \{z\}) - g(T^*) - w(T^*) + |T^*|(w_z + g_z)
 \end{aligned}$$

- Rearranging terms: $w_z + g_z < \frac{w(T^*) + g(T^*)}{|T^*|}$ ■

* 7.13 Assignment Problem

Assignment Problem

Assignment problem.

- Input: **weighted**, complete bipartite graph $G = (L \cup R, E)$ with $|L| = |R|$.
- Goal: find a perfect matching of **min weight**.

	1'	2'	3'	4'	5'
1	3	8	9	15	10
2	4	10	7	16	14
3	9	13	11	19	10
4	8	13	12	20	13
5	1	7	5	11	9

Min cost perfect matching

$M = \{ 1-2', 2-3', 3-5', 4-1', 5-4' \}$

$\text{cost}(M) = 8 + 7 + 10 + 8 + 11 = 44$

Applications

Natural applications.

- Match jobs to machines.
- Match personnel to tasks.
- Match PU students to writing seminars.

Non-obvious applications.

- Vehicle routing.
- Signal processing.
- Virtual output queueing.
- Multiple object tracking.
- Approximate string matching.
- Enhance accuracy of solving linear systems of equations.

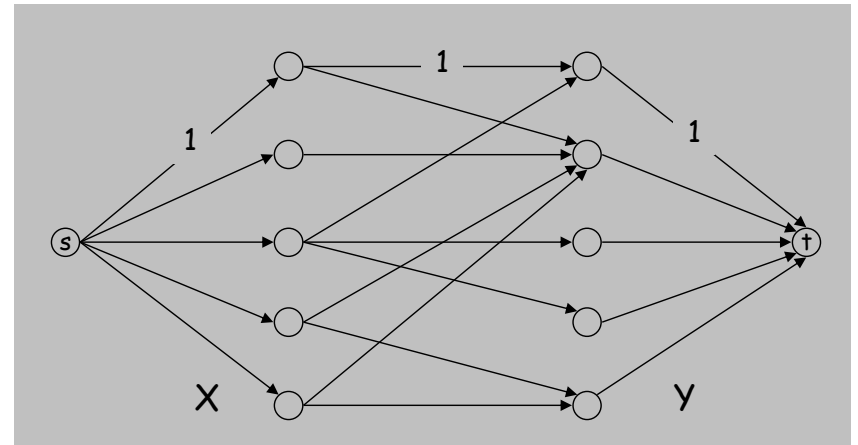
Bipartite Matching

Bipartite matching. Can solve via reduction to max flow.

Flow. During Ford-Fulkerson, all capacities and flows are 0/1. Flow corresponds to edges in a matching M .

Residual graph G_M simplifies to:

- If $(x, y) \notin M$, then (x, y) is in G_M .
- If $(x, y) \in M$, the (y, x) is in G_M .

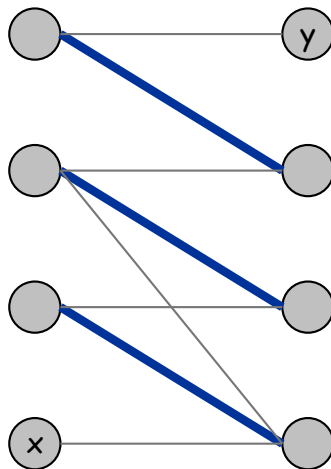


Augmenting path simplifies to:

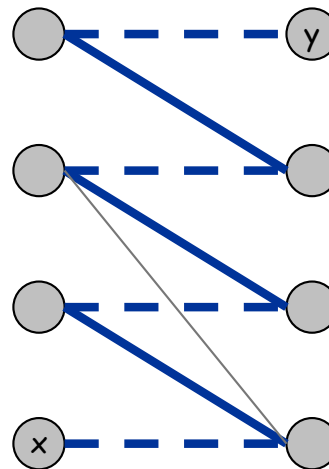
- Edge from s to an unmatched node $x \in X$.
- Alternating sequence of unmatched and matched edges.
- Edge from unmatched node $y \in Y$ to t .

Alternating Path

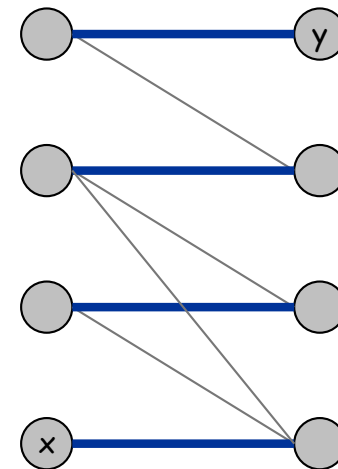
Alternating path. Alternating sequence of unmatched and matched edges, from unmatched node $x \in X$ to unmatched node $y \in Y$.



matching M



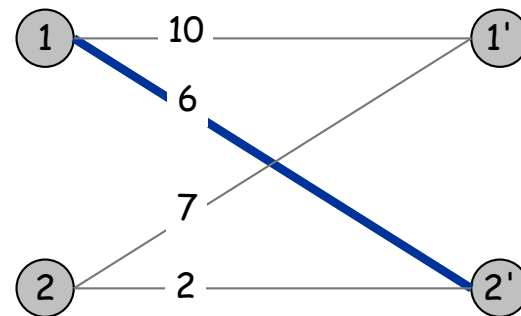
alternating path



matching M'

Assignment Problem: Successive Shortest Path Algorithm

Cost of an alternating path. Pay $c(x, y)$ to match $x-y$; receive $c(x, y)$ to unmatch $x-y$.



$$\begin{aligned}\text{cost}(2 - 1') &= 7 \\ \text{cost}(2 - 2' - 1 - 1') &= 2 - 6 + 10 = 6\end{aligned}$$

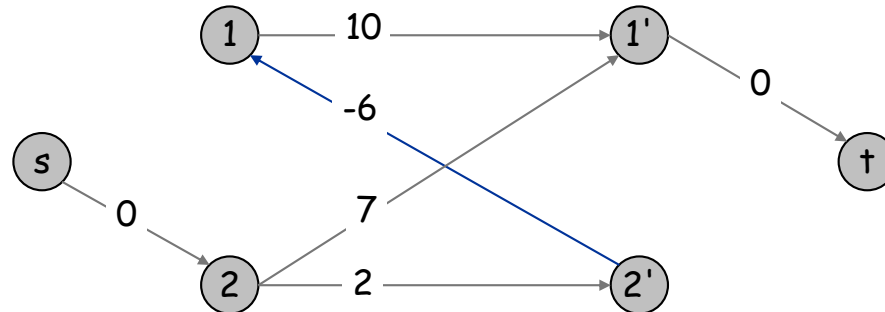
Shortest alternating path. Alternating path from any unmatched node $x \in X$ to any unmatched node $y \in Y$ with smallest cost.

Successive shortest path algorithm.

- Start with empty matching.
- Repeatedly augment along a **shortest** alternating path.

Finding The Shortest Alternating Path

Shortest alternating path. Corresponds to shortest s - t path in G_M .



Concern. Edge costs can be negative.

Fact. If always choose shortest alternating path, then G_M contains no negative cycles \Rightarrow compute using Bellman-Ford.

Our plan. Use **duality** to avoid negative edge costs (and negative cost cycles) \Rightarrow compute using Dijkstra.

Equivalent Assignment Problem

Duality intuition. Adding (or subtracting) a constant to every entry in row x or column y does not change the min cost perfect matching(s).

$c(x, y)$						$c^p(x, y)$				
3	8	9	15	10	subtract 11 from column 4 →	3	8	9	4	10
4	10	7	16	14		4	10	7	2	14
9	13	11	19	10		9	13	11	8	10
8	13	12	20	13		8	13	12	9	13
1	7	5	11	9		1	7	5	0	9
11										

Equivalent Assignment Problem

Duality intuition. Adding $p(x)$ to row x and subtracting $p(y)$ from row y does not change the min cost perfect matching(s).

$c(x, y)$							$c^p(x, y)$					
3	8	9	15	10	5	→	0	0	3	1	2	
4	10	7	16	14	4		0	1	0	1	5	
9	13	11	19	10	3		4	3	3	3	0	
8	13	12	20	13	0		0	0	1	1	0	
1	7	5	11	9	8		1	2	2	0	4	
8	13	11	19	13								↑ $9 + 8 - 13$

Reduced Costs

Reduced costs. For $x \in X, y \in Y$, define $c^p(x, y) = p(x) + c(x, y) - p(y)$.

Observation 1. Finding a min cost perfect matching with reduced costs is equivalent to finding a min cost perfect matching with original costs.

$c(x, y)$							$c^p(x, y)$				
3	8	9	15	10	5	→	0	0	3	1	2
4	10	7	16	14	4		0	1	0	1	5
9	13	11	19	10	3		4	3	3	3	0
8	13	12	20	13	0		0	0	1	1	0
1	7	5	11	9	8		1	2	2	0	4
8	13	11	19	13							
											↑ $9 + 8 - 13$

Compatible Prices

Compatible prices. For each node v , maintain prices $p(v)$ such that:

- (i) $c^p(x, y) \geq 0$ for for all $(x, y) \notin M$.
- (ii) $c^p(x, y) = 0$ for for all $(x, y) \in M$.

Observation 2. If p are compatible prices for a **perfect** matching M , then M is a min cost perfect matching.

$c(x, y)$							$c^p(x, y)$				
3	8	9	15	10	5	→	0	0	3	1	2
4	10	7	16	14	4		0	1	0	1	5
9	13	11	19	10	3		4	3	3	3	0
8	13	12	20	13	0		0	0	1	1	0
1	7	5	11	9	8		1	2	2	0	4
8	13	11	19	13							

$$\text{cost}(M) = \sum_{(x, y) \in M} c(x, y) = (8+7+10+8+11) = 44$$

$$\text{cost}(M) = \sum_{y \in Y} p(y) - \sum_{x \in X} p(x) = (8+13+11+19+13) - (5+4+3+0+8) = 44$$

Successive Shortest Path Algorithm

Successive shortest path.

```
Successive-Shortest-Path( $X, Y, c$ ) {  
   $M \leftarrow \phi$   
  foreach  $x \in X$ :  $p(x) \leftarrow 0$   
  foreach  $y \in Y$ :  $p(y) \leftarrow \min_{e \text{ into } y} c(e)$  p is compatible  
with  $M = \phi$   
  
  while (M is not a perfect matching) {  
    Compute shortest path distances  $d$   
     $P \leftarrow$  shortest alternating path using costs  $c^P$   
     $M \leftarrow$  updated matching after augmenting along  $P$   
    foreach  $v \in X \cup Y$ :  $p(v) \leftarrow p(v) + d(v)$   
  }  
  return  $M$   
}
```

Maintaining Compatible Prices

Lemma 1. Let p be compatible prices for matching M . Let d be shortest path distances in G_M with costs c^p . All edges (x, y) on shortest path have $c^{p+d}(x, y) = 0$.

↑
forward or reverse edges

Pf. Let (x, y) be some edge on shortest path.

- If $(x, y) \in M$, then (y, x) on shortest path and $d(x) = d(y) - c^p(x, y)$.
If $(x, y) \notin M$, then (x, y) on shortest path and $d(y) = d(x) + c^p(x, y)$.
- In either case, $d(x) + c^p(x, y) - d(y) = 0$.
- By definition, $c^p(x, y) = p(x) + c(x, y) - p(y)$.
- Substituting for $c^p(x, y)$ yields:
 $(p(x) + d(x)) + c(x, y) - (p(y) + d(y)) = 0$.
- In other words, $c^{p+d}(x, y) = 0$. ■

Reduced costs: $c^p(x, y) = p(x) + c(x, y) - p(y)$.

Maintaining Compatible Prices

Lemma 2. Let p be compatible prices for matching M . Let d be shortest path distances in G_M with costs c^p . Then $p' = p + d$ are also compatible prices for M .

Pf. $(x, y) \in M$

- (y, x) is the only edge entering x in G_M . Thus, (y, x) on shortest path.
- By Lemma 1, $c^{p+d}(x, y) = 0$.

Pf. $(x, y) \notin M$

- (x, y) is an edge in $G_M \Rightarrow d(y) \leq d(x) + c^p(x, y)$.
- Substituting $c^p(x, y) = p(x) + c(x, y) - p(y) \geq 0$ yields $(p(x) + d(x)) + c(x, y) - (p(y) + d(y)) \geq 0$.
- In other words, $c^{p+d}(x, y) \geq 0$. ■

Compatible prices. For each node v :

- (i) $c^p(x, y) \geq 0$ for for all $(x, y) \notin M$.
- (ii) $c^p(x, y) = 0$ for for all $(x, y) \in M$.

Maintaining Compatible Prices

Lemma 3. Let M' be matching obtained by augmenting along a min cost path with respect to c^{p+d} . Then $p' = p + d$ is compatible with M' .

Pf.

- By Lemma 2, the prices $p + d$ are compatible for M .
- Since we augment along a min cost path, the only edges (x, y) that swap into or out of the matching are on the shortest path.
- By Lemma 1, these edges satisfy $c^{p+d}(x, y) = 0$.
- Thus, compatibility is maintained. ■

Compatible prices. For each node v :

- (i) $c^p(x, y) \geq 0$ for for all $(x, y) \notin M$.
- (ii) $c^p(x, y) = 0$ for for all $(x, y) \in M$.

Successive Shortest Path: Analysis

Invariant. The algorithm maintains a matching M and compatible prices p .

Pf. Follows from Lemmas 2 and 3 and initial choice of prices. ■

Theorem. The algorithm returns a min cost perfect matching.


Pf. Upon termination M is a perfect matching, and p are compatible prices. Optimality follows from Observation 2. ■

Theorem. The algorithm can be implemented in $O(n^3)$ time.

Pf.

- Each iteration increases the cardinality of M by 1 $\Rightarrow n$ iterations.
- Bottleneck operation is computing shortest path distances d .
Since all costs are nonnegative, each iteration takes $O(n^2)$ time using (dense) Dijkstra. ■

Weighted Bipartite Matching

Weighted bipartite matching. Given weighted bipartite graph, find maximum cardinality matching of minimum weight.  m edges, n nodes

Successive shortest path algorithm. $O(mn \log n)$ time using heap-based version of Dijkstra's algorithm.

Best known bounds. $O(mn^{1/2})$ deterministic; $O(n^{2.376})$ randomized.

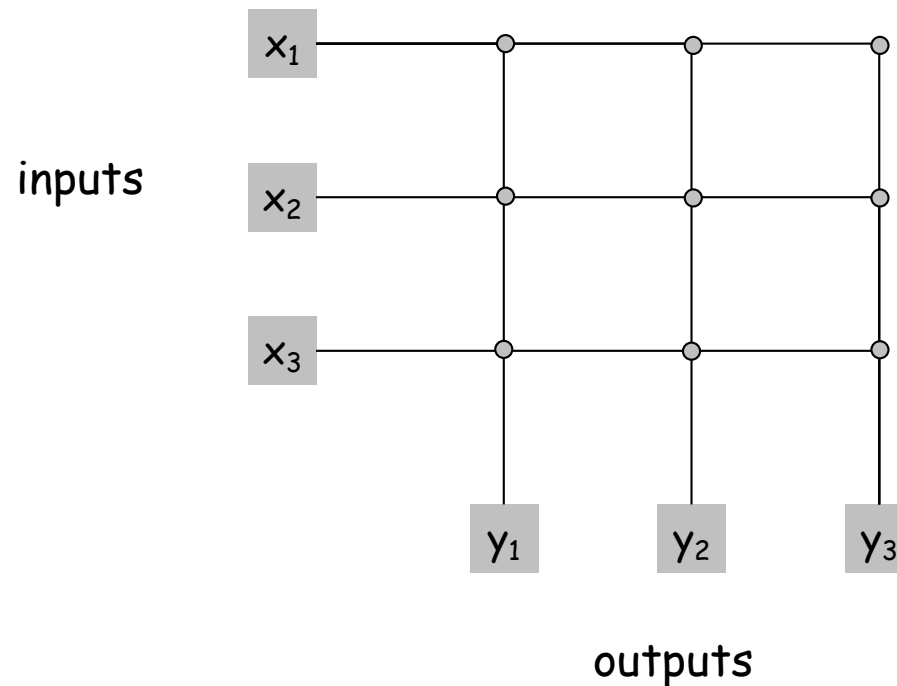
Planar weighted bipartite matching. $O(n^{3/2} \log^5 n)$.

Input Queued Switching

Input-Queued Switching

Input-queued switch.

- n inputs and n outputs in an n -by- n crossbar layout.
- At most one cell can depart an input at a time.
- At most one cell can arrive at an output at a time.
- Cell arrives at input x and must be routed to output y .

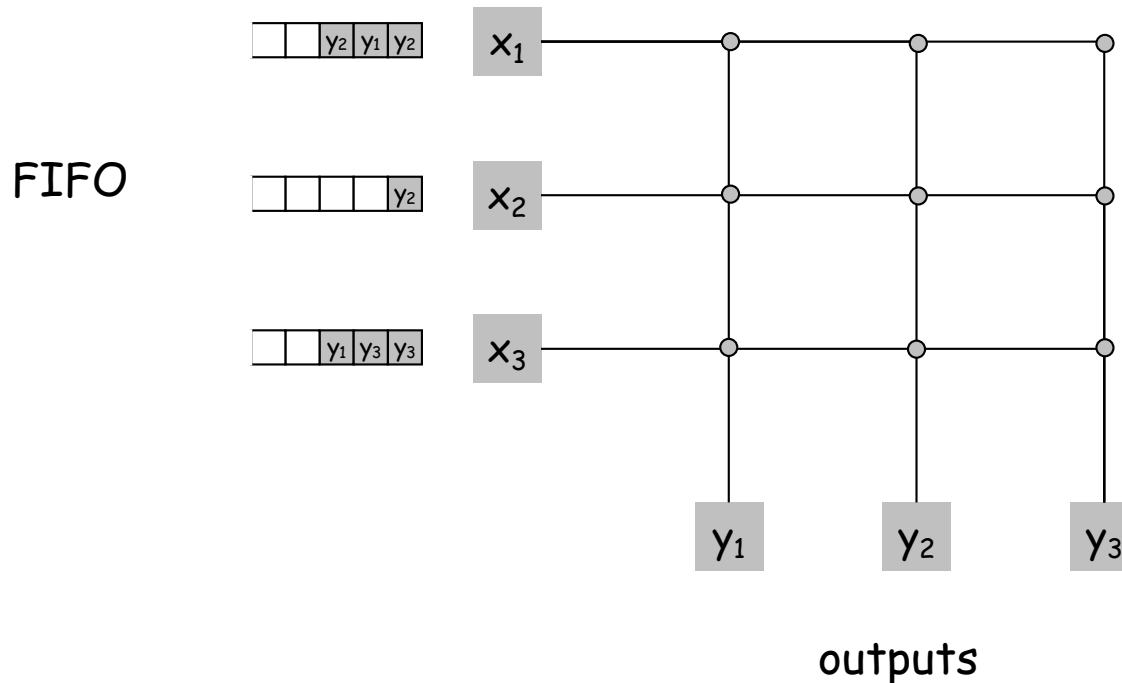


Input-Queued Switching

FIFO queueing. Each input x maintains one queue of cells to be routed.

Head-of-line blocking (HOL).

- A cell can be blocked by a cell queued ahead of it that is destined for a different output.
- Can limit throughput to 58%, even when arrivals are uniform.

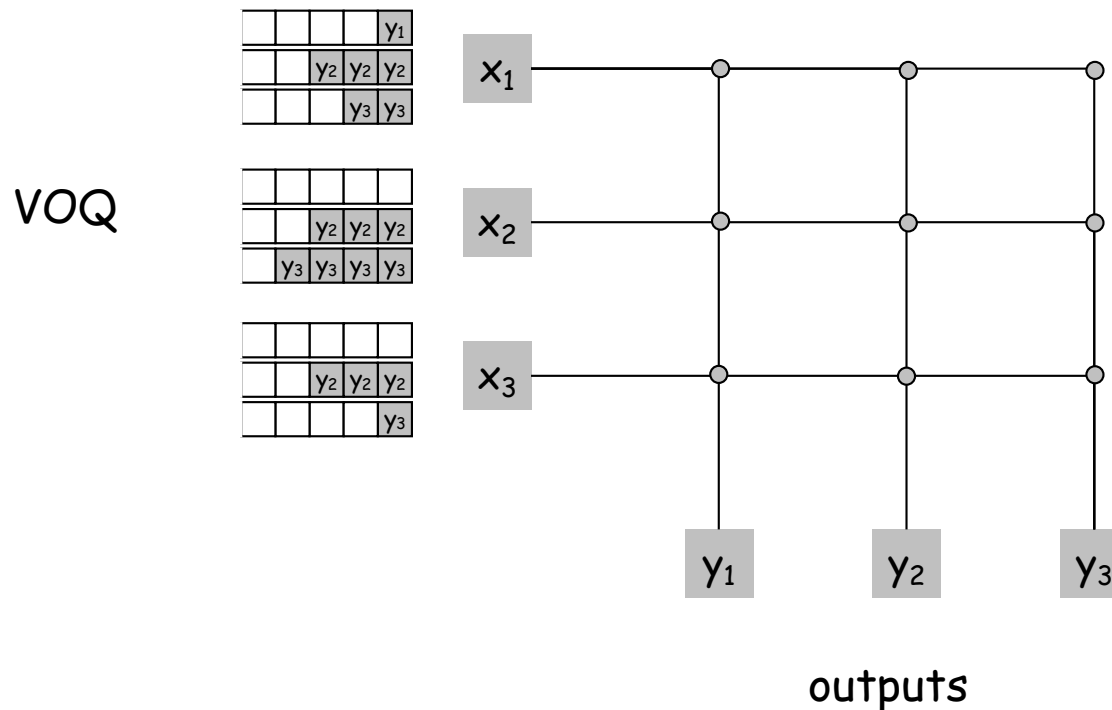


Input-Queued Switching

Virtual output queueing (VOQ). Each input x maintains n queue of cells, one for each output y .

Maximum size matching. Find a max cardinality matching.

- Achieves 100% when arrivals are uniform.
- Can starve input-queues when arrivals are non-uniform.



Input-Queued Switching

Max weight matching. Find a min cost perfect matching between inputs x and outputs y , where $c(x, y)$ equals:

- [LQF] The number of cells waiting to go from input x to output y .
- [OCF] The waiting time of the cell at the head of VOQ from x to y .

Theorem. LQF and OCF achieve 100% throughput if arrivals are independent.

Practice.

- Too slow in practice for this application; difficult to implement in hardware. Provides theoretical framework.
- Use **maximal** (weighted) matching \Rightarrow 2-approximation.