



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Instructor: Shan CHEN (陈杉)

chens3@sustech.edu.cn

(slides edited from Prof. Shiqi Yu)



南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Network Flow



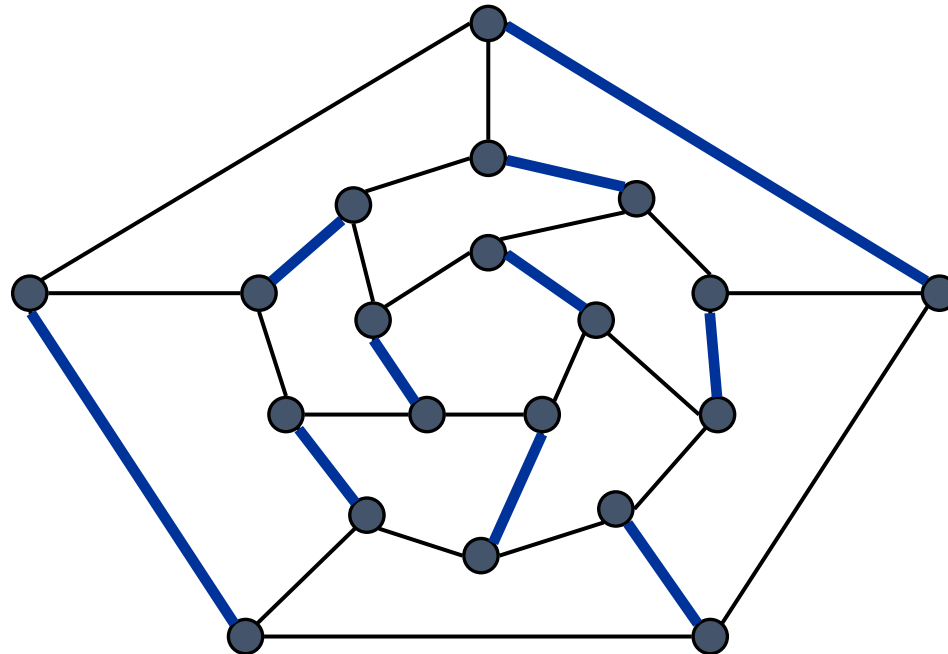
7. Bipartite Matching



Matching

- **Matching.**

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

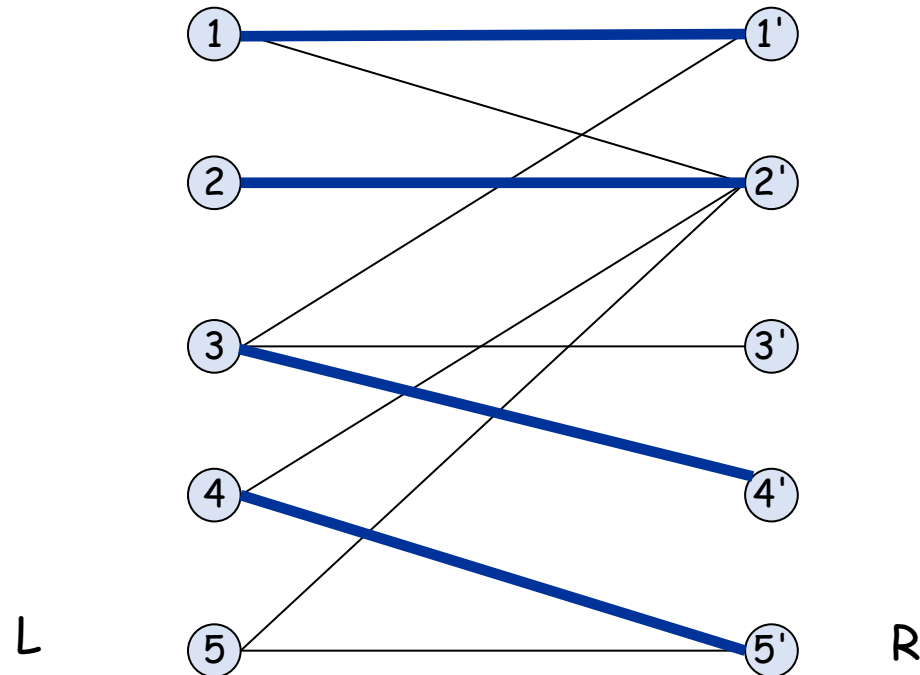




Bipartite Matching

- **Bipartite matching.**

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

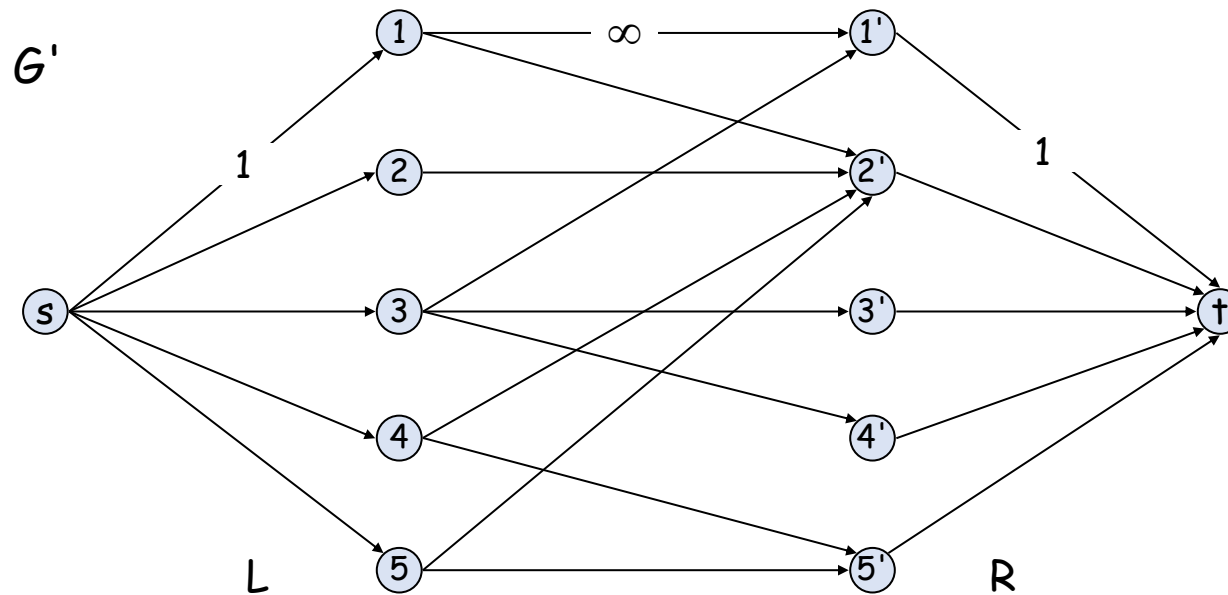




Bipartite Matching: Max-Flow Formulation

- **Max-flow formulation.**

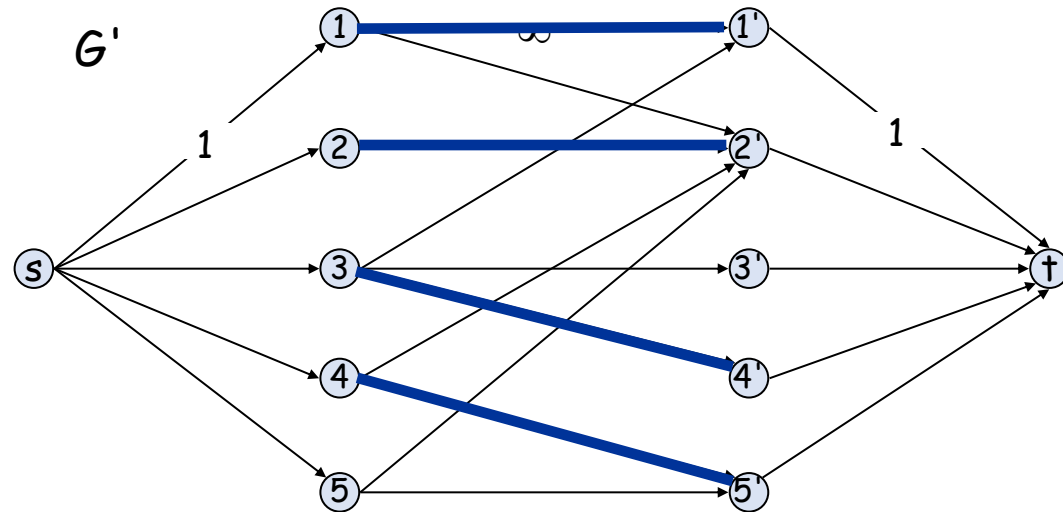
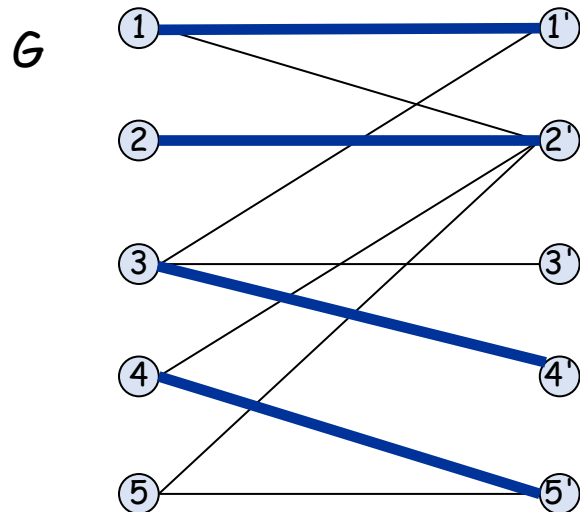
- Create digraph $G' = (L \cup R \cup \{s, t\}, E')$.
- Direct all edges from L to R , and assign infinite (or unit) capacity.
- Add source s , and unit capacity edges from s to each node in L .
- Add sink t , and unit capacity edges from each node in R to t .





Max-Flow Formulation: Correctness

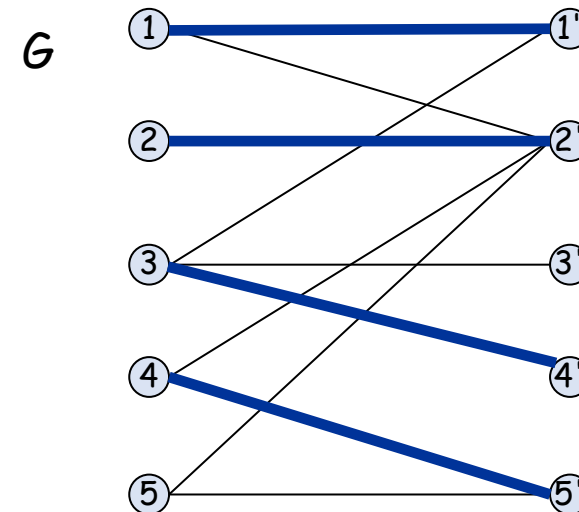
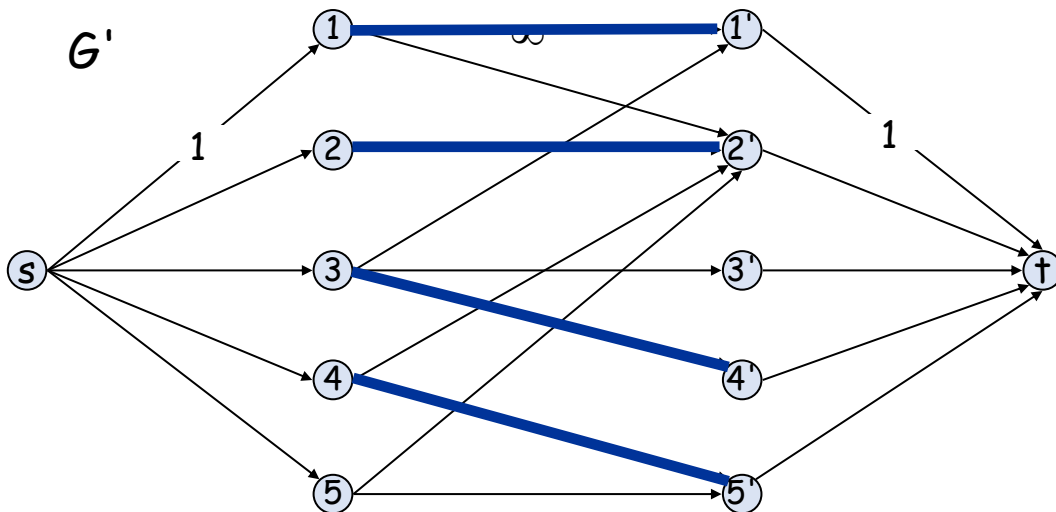
- **Theorem.** 1-1 correspondence between matchings of cardinality k in G and integral flows of value k in G' .
- **Pf.** \Rightarrow : Let M be a matching in G of cardinality k .
 - Consider flow f that sends 1 unit on each of the k corresponding paths.
 - f is an integral flow of value k . ▀





Max-Flow Formulation: Correctness

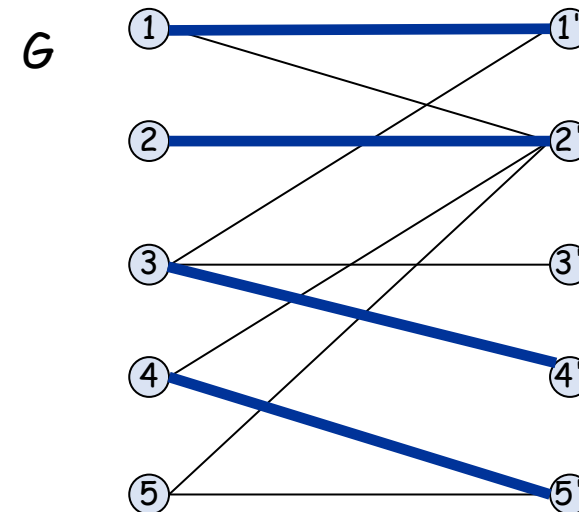
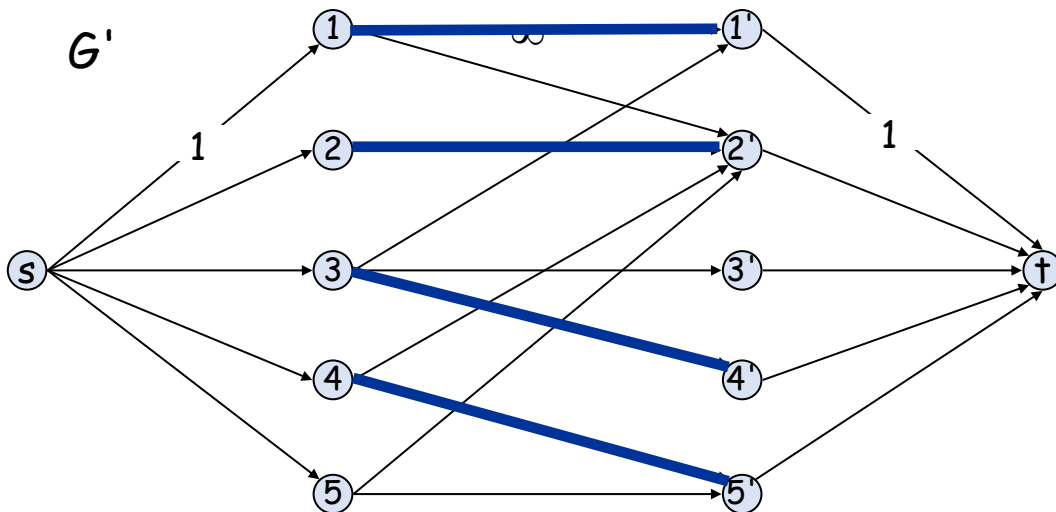
- **Theorem.** 1-1 correspondence between matchings of cardinality k in G and integral flows of value k in G' .
- **Pf.** \Leftarrow : Let f be an integral flow in G' of value k .
 - Consider M = set of edges from L to R with $f(e) = 1$.
 - ✓ each node in L and R participates in at most one edge in M
 - ✓ $|M| = k$: apply flow-value lemma to cut $(L \cup \{s\}, R \cup \{t\})$ ■





Max-Flow Formulation: Correctness

- **Theorem.** 1-1 correspondence between matchings of cardinality k in G and integral flows of value k in G' .
- **Corollary.** Can solve bipartite matching via max-flow formulation.
Pf.
 - Integrality theorem \Rightarrow there exists a max flow f^* in G' that is integral.
 - Theorem $\Rightarrow f^*$ corresponds to max-cardinality matching. ■





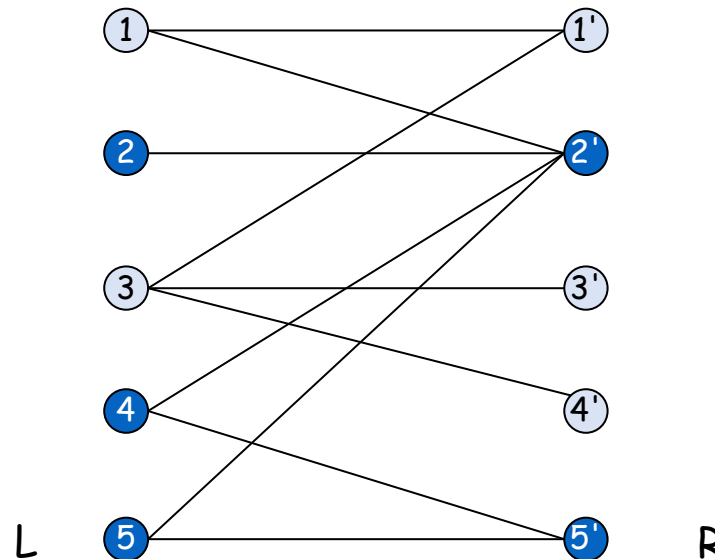
Perfect Matchings in Bipartite Graphs

- **Def.** Given a graph $G = (V, E)$, a subset of edges $M \subseteq E$ is a **perfect matching** 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 other conditions are sufficient?



Perfect Matchings in Bipartite Graphs

- **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 $N(S)$.

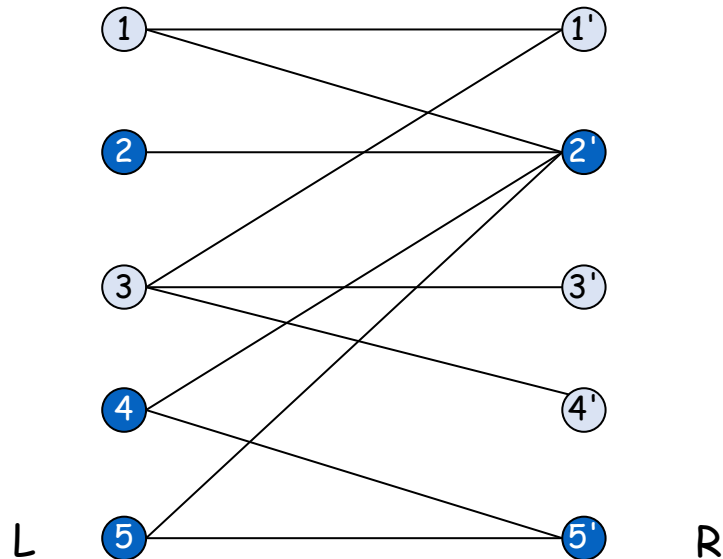


No perfect matching:
 $S = \{ 2, 4, 5 \}$
 $N(S) = \{ 2', 5' \}.$



Hall's Marriage Theorem

- **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.



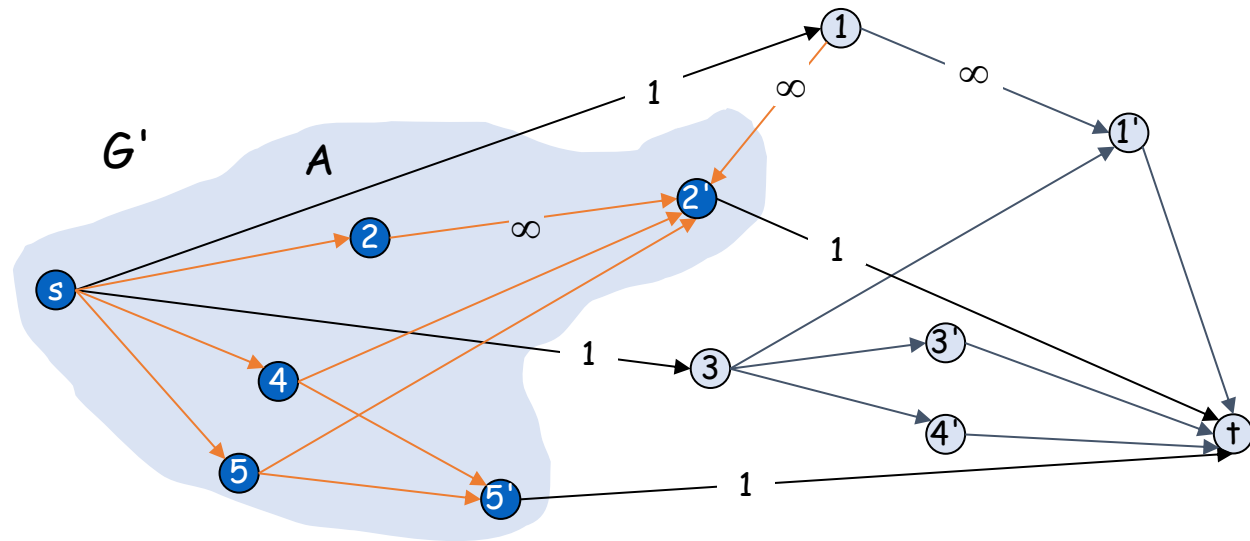
No perfect matching:
 $S = \{ 2, 4, 5 \}$
 $N(S) = \{ 2', 5' \}.$



Hall's Marriage Theorem

- **Pf.** \Leftarrow : Suppose G does not have a perfect matching. (**contrapositive**)
 - Formulate as a max flow problem and let (A, B) be a min cut in G' .
 - By max-flow min-cut theorem, $c(A, B) < |L|$.
 - Define $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$.
 - **Min cut cannot use ∞ edges:** $N(L_A) \subseteq R \cap A = R_A$
 - $c(A, B) = |L_B| + |R_A| < |L| = |L_A| + |L_B| \Rightarrow |R_A| < |L_A|$
 - $|N(L_A)| \leq |R_A| < |L_A|$.
 - Choose $S = L_A$. ▀

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





Algorithms for Matching

- **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: $O(mn^{1/2})$.
 - Fast matrix multiplication: $O(n^{2.378})$. [Mucha-Sankowski 2003]
- **Non-bipartite matching.**
 - Structure of non-bipartite (undirected) graphs is more complicated.
 - But well-understood. [Tutte-Berge formula, Edmonds-Galai]
 - Blossom algorithm: $O(n^4)$. [Edmonds 1965]
 - Best known: $O(mn^{1/2})$. [Micali-Vazirani 1980, Vazirani 1994]



Historical Significance (Jack Edmonds 1965)

2. Digression. An explanation is due on the use of the words “efficient algorithm.” First, what I present is a conceptual description of an algorithm and not a particular formalized algorithm or “code.”

For practical purposes computational details are vital. However, my purpose is only to show as attractively as I can that there is an efficient algorithm. According to the dictionary, “efficient” means “adequate in operation or performance.” This is roughly the meaning I want—in the sense that it is conceivable for maximum matching to have no efficient algorithm. Perhaps a better word is “good.”

I am claiming, as a mathematical result, the existence of a *good* algorithm for finding a maximum cardinality matching in a graph.

There is an obvious finite algorithm, but that algorithm increases in difficulty exponentially with the size of the graph. It is by no means obvious whether *or not* there exists an algorithm whose difficulty increases only algebraically with the size of the graph.

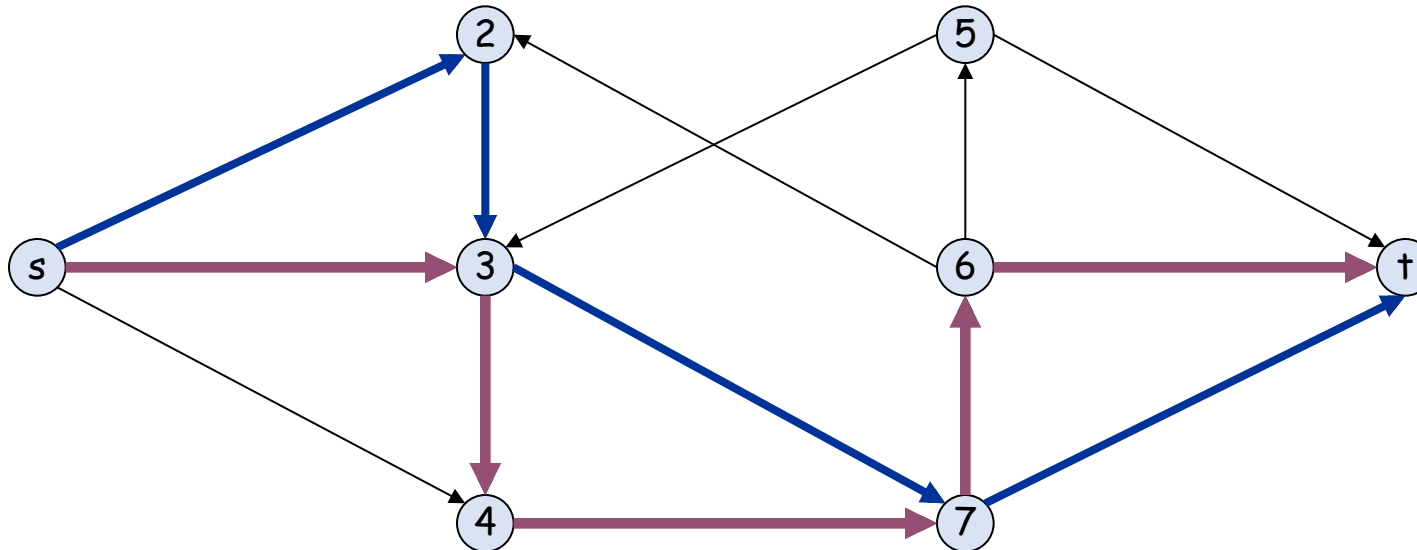


8. Disjoint Paths



Edge-Disjoint Paths

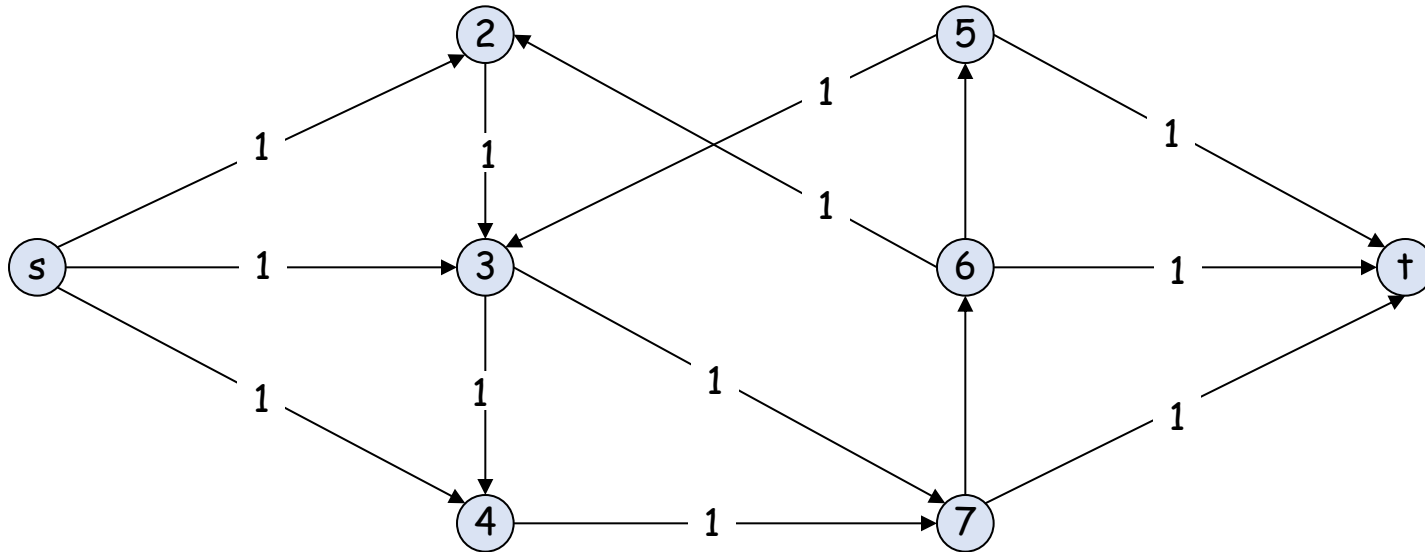
- **Def.** Two paths are **edge-disjoint** if they have no edge in common.
- **Edge-disjoint paths problem.** Given a digraph $G = (V, E)$ and two nodes s and t , find the max number of edge-disjoint s - t paths.
- **Ex.** Communication networks.





Edge-Disjoint Paths: Max-Flow Formulation

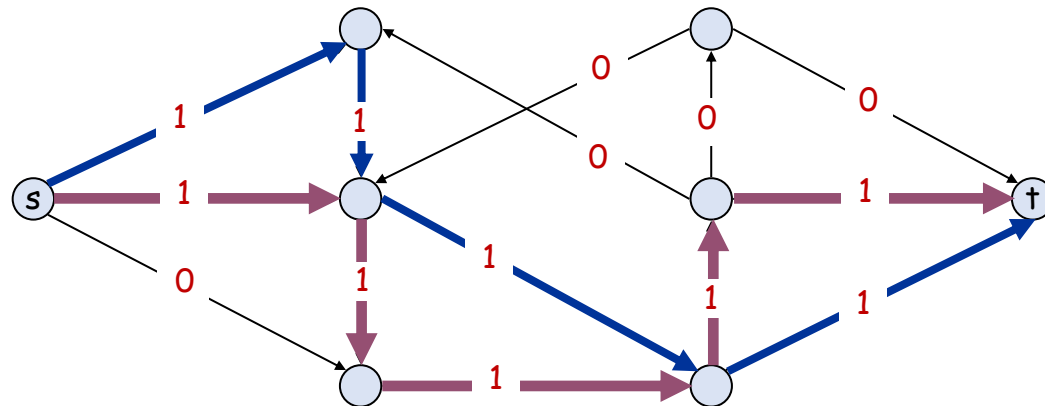
- **Def.** Two paths are **edge-disjoint** if they have no edge in common.
- **Edge-disjoint paths problem.** Given a digraph $G = (V, E)$ and two nodes s and t , find the max number of edge-disjoint s - t paths.
- **Max-flow formulation.** Assign **unit capacity** to every edge.





Max-Flow Formulation: Correctness

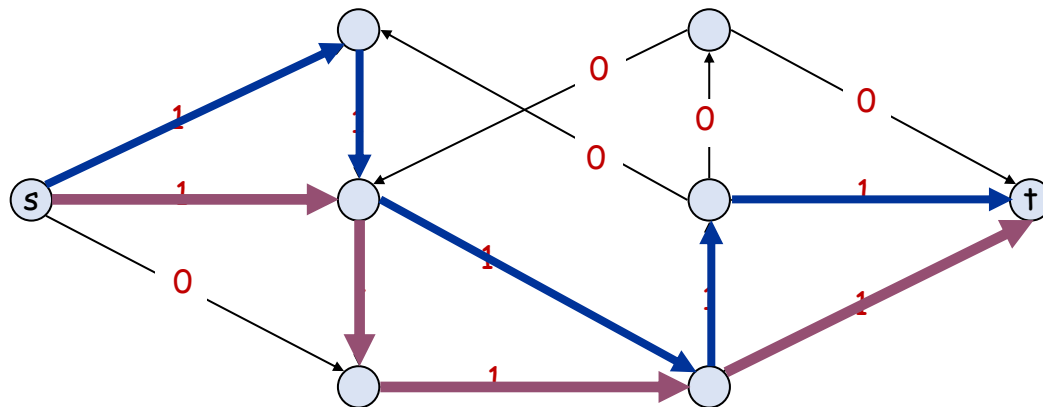
- **Theorem.** 1-1 correspondence between k edge-disjoint s - t paths in G and integral flows of value k in G' .
- **Pf.** \Rightarrow :
 - Let P_1, \dots, P_k be k edge-disjoint paths in G .
 - Set $f(e) = 1$ if edge e participates in some path P_i ; else set $f(e) = 0$.
 - Since paths are edge-disjoint, f is a flow of value k . ▀





Max-Flow Formulation: Correctness

- **Theorem.** 1-1 correspondence between k edge-disjoint s - t paths in G and integral flows of value k in G' .
- **Pf.** \Leftarrow :
 - Let f be an integral flow in G' of value k .
 - Consider edge (s, u) with $f(s, u) = 1$.
 - ✓ by flow 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. ■

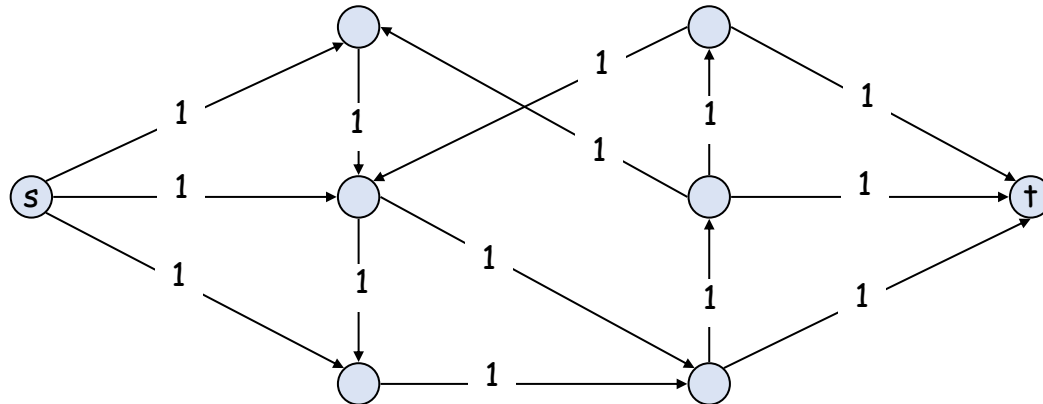


can eliminate cycles and get simple paths if desired



Max-Flow Formulation: Correctness

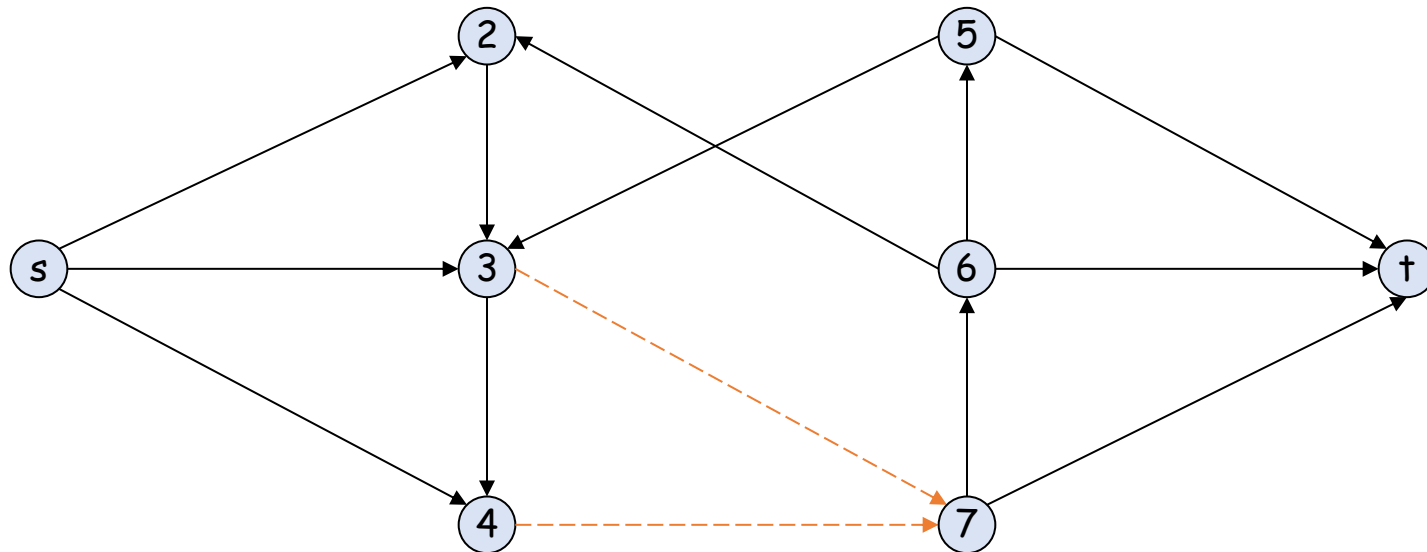
- **Max-flow formulation.** Assign unit capacity to every edge.
- **Theorem.** 1-1 correspondence between k edge-disjoint s - t paths in G and integral flows of value k in G' .
- **Corollary.** Can solve edge-disjoint paths via max-flow formulation.
- **Pf.**
 - Integrality theorem \Rightarrow there exists a max flow f^* in G' that is integral.
 - Theorem $\Rightarrow f^*$ corresponds to max number of edge-disjoint s - t paths in G . ■





Network Connectivity

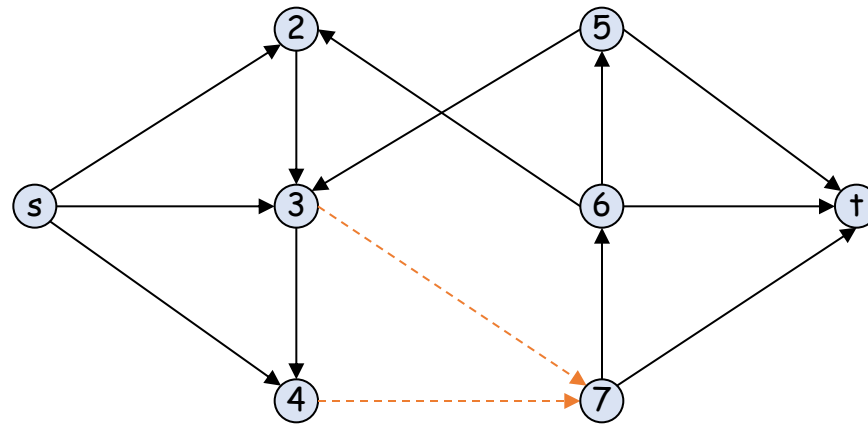
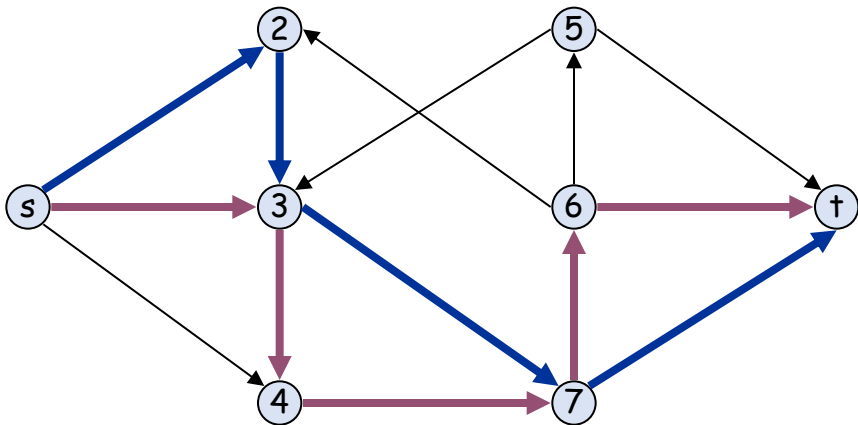
- **Def.** A set of edges $F \subseteq E$ **disconnects t from s** if every s - t path uses at least one edge in F .
- **Network connectivity.** Given a digraph $G = (V, E)$ and two nodes s and t , find min number of edges whose removal disconnects t from s .





Menger's Theorem

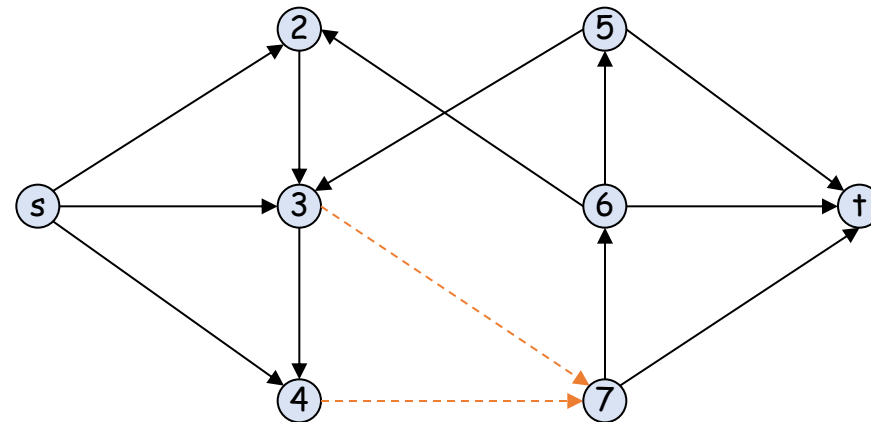
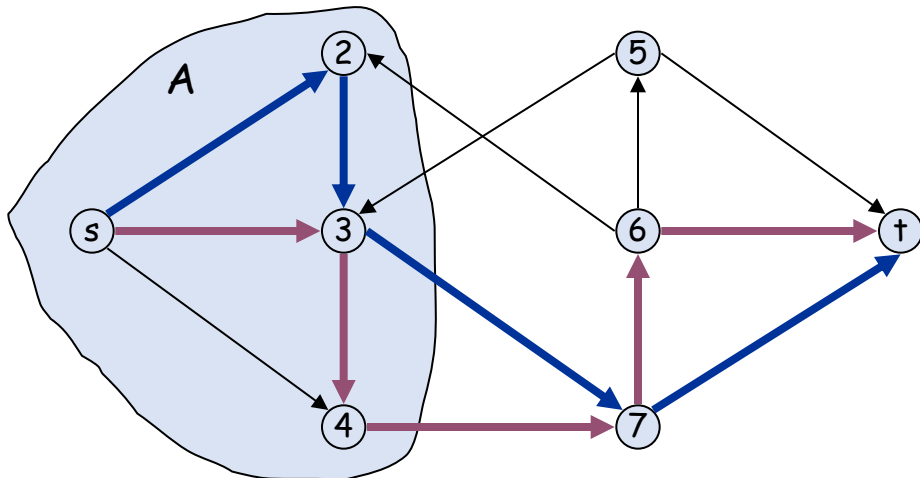
- **Theorem.** [Menger 1927] The max number of edge-disjoint s - t paths equals the min number of edges whose removal disconnects t from s .
- **Pf.** \leq :
 - Suppose the removal of $F \subseteq E$ disconnects t from s , and $|F| = k$.
 - Every s - t path uses at least one edge in F .
 - Hence, the number of edge-disjoint paths is $\leq k$. ▀





Menger's Theorem

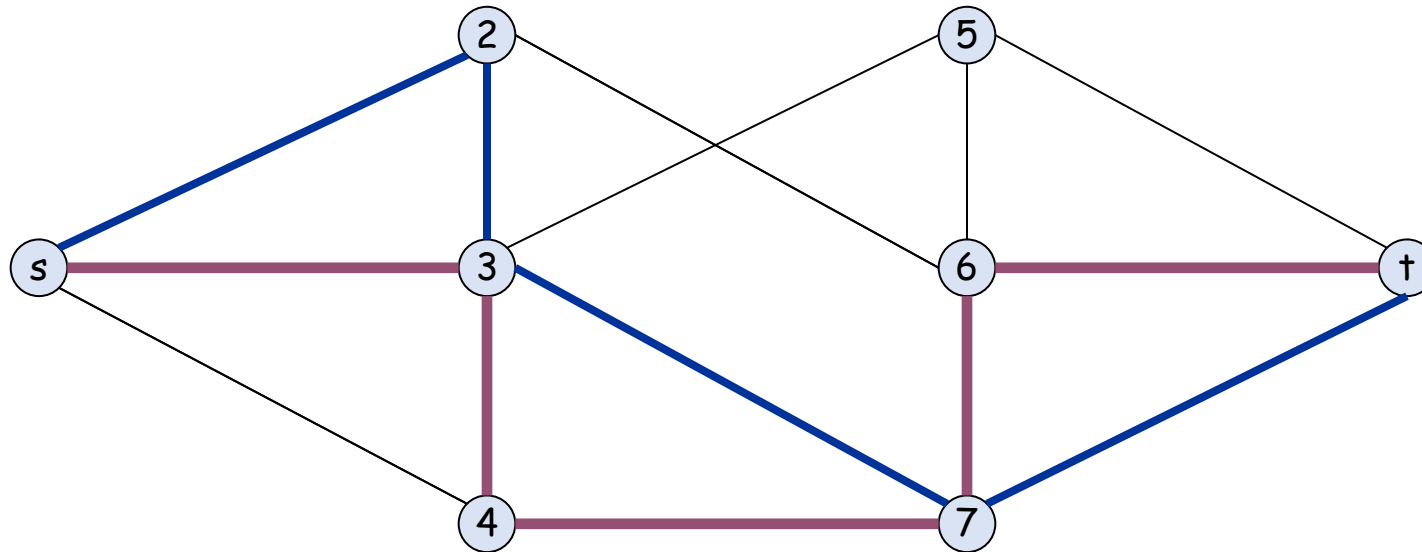
- **Theorem.** [Menger 1927] The max number of edge-disjoint s - t paths equals the min number of edges whose removal disconnects t from s .
- **Pf.** \geq :
 - Suppose max number of edge-disjoint s - t paths is k . Then, value of max flow = k .
 - Max-flow min-cut theorem \Rightarrow there exists a cut (A, B) of capacity k .
 - Let F be set of edges going from A to B .
 - $|F| = k$ and removing F disconnects t from s . ■





Edge-Disjoint Paths in Undirected Graphs

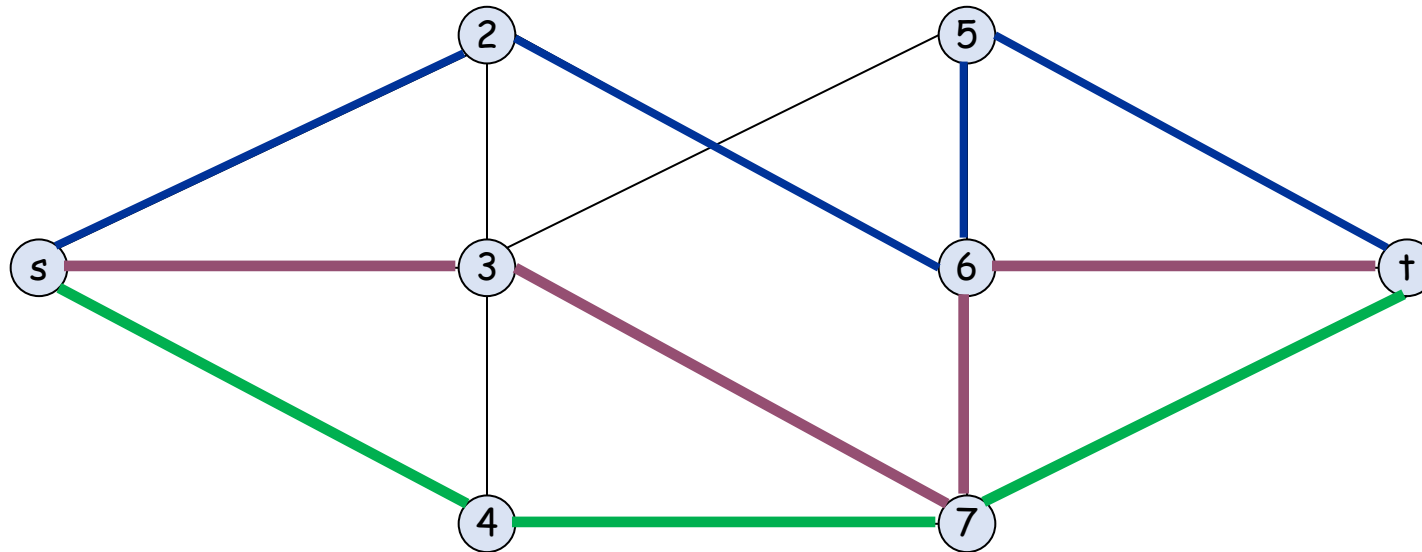
- **Edge-disjoint paths problem in undirected graphs.** Given an (undirected) graph $G = (V, E)$ and two nodes s and t , find the max number of edge-disjoint s - t paths.
- **Ex.** 2 edge-disjoint paths.





Edge-Disjoint Paths in Undirected Graphs

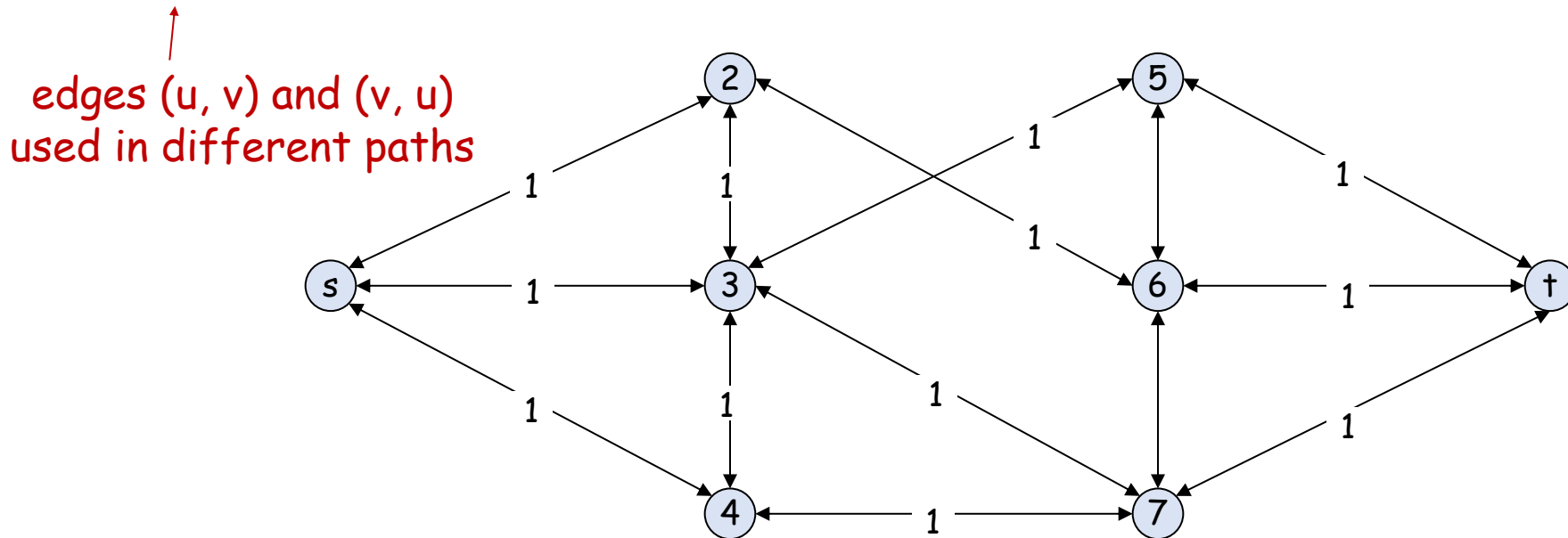
- **Edge-disjoint paths problem in undirected graphs.** Given an (undirected) graph $G = (V, E)$ and two nodes s and t , find the max number of edge-disjoint s - t paths.
- **Ex.** 3 edge-disjoint paths (max number).





Edge-Disjoint Paths: Max-Flow Formulation

- **Max-flow formulation.** Replace each edge with **two antiparallel edges** and assign unit capacity to every edge.
- **Observation.** Two paths P_1 and P_2 may be edge-disjoint in the digraph but not edge-disjoint in the undirected graph.





Max-Flow Formulation: Correctness

- **Max-flow formulation.** Replace each edge with **two antiparallel edges** and assign **unit capacity** to every edge.
- **Lemma.** In any flow network, there exists a maximum flow f in which for each pair of antiparallel edges e and e' : either $f(e) = 0$ or $f(e') = 0$ or both. Moreover, integrality theorem still holds.
- **Pf. (by induction on number of such pairs)**
 - Suppose $f(e) > 0$ and $f(e') > 0$ for a pair of antiparallel edges e and e' .
 - Set $f(e) = f(e) - \delta$ and $f(e') = f(e') - \delta$, where $\delta = \min \{f(e), f(e')\}$.
 - f is still a flow of the same value but has one fewer such pair. ▀



Max-Flow Formulation: Correctness

- **Max-flow formulation.** Replace each edge with **two antiparallel edges** and assign unit capacity to every edge.
- **Lemma.** In any flow network, there exists a maximum flow f in which for each pair of antiparallel edges e and e' : either $f(e) = 0$ or $f(e') = 0$ or both. Moreover, integrality theorem still holds.
- **Theorem.** Max number of edge-disjoint s - t paths = value of max flow.
- **Pf.** Similar to proof in digraphs; use Lemma.



More Menger Theorems

- **Theorem.** Given an **undirected** graph and two nodes s and t , the max number of **edge-disjoint** s – t paths equals the min number of edges whose removal disconnects s and t .
- **Theorem.** Given an **undirected** graph and two nonadjacent nodes s and t , the max number of internally **node-disjoint** s – t paths equals the min number of internal nodes whose removal disconnects s and t .
- **Theorem.** Given a **directed** graph with two nonadjacent nodes s and t , the max number of internally **node-disjoint** s – t paths equals the min number of internal nodes whose removal disconnects t from s .



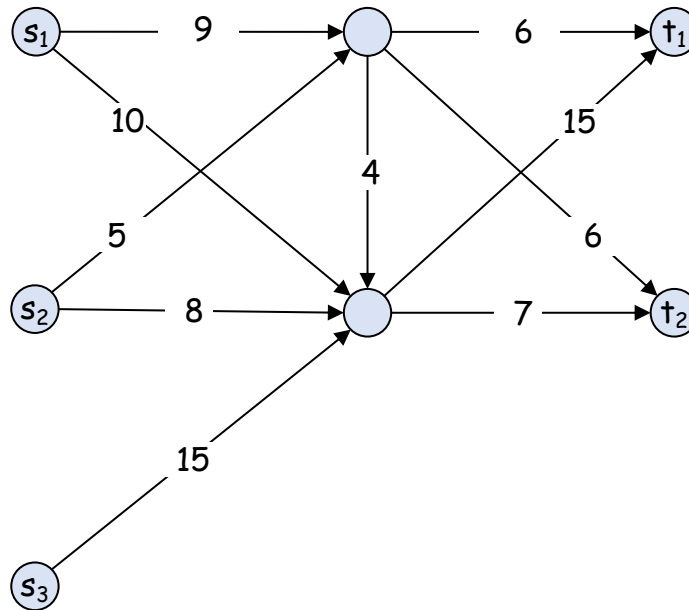
9. Extensions to Max Flow



Multiple Sources and Sinks

- **Def.** Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$ and multiple source nodes and multiple sink nodes, find max flow that can be sent from the source nodes to the sink nodes.

flow network G

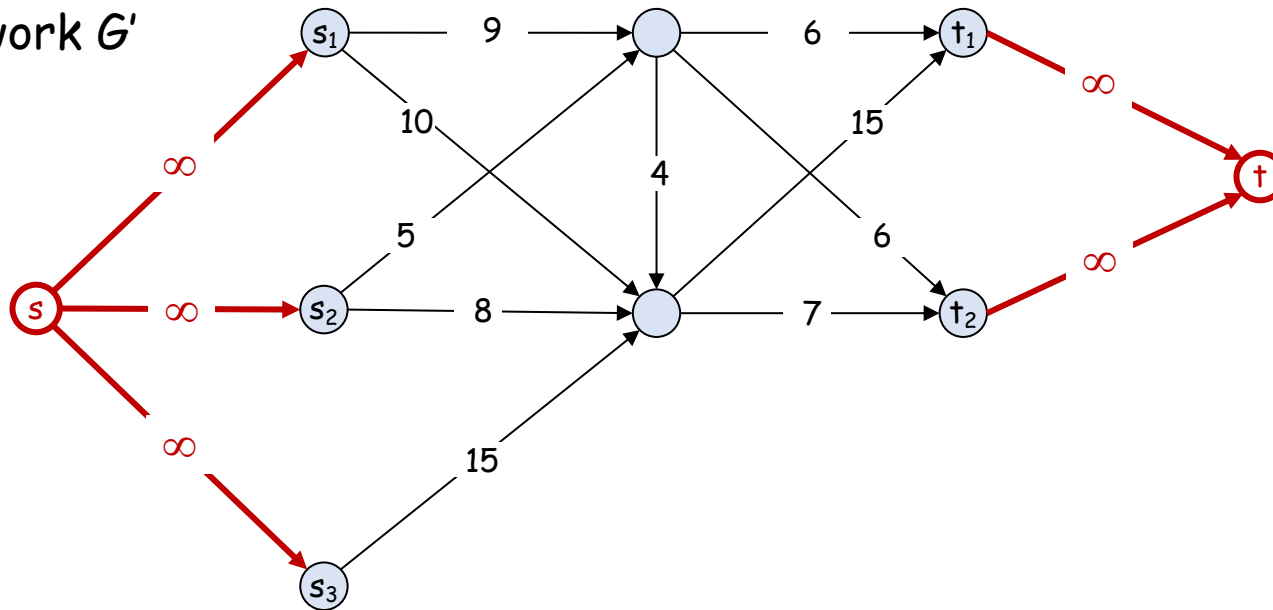




Multiple Sources and Sinks

- **Def.** Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$ and multiple source nodes and multiple sink nodes, find max flow that can be sent from the source nodes to the sink nodes.
- **Claim.** 1-1 correspondence between flows in G and G' .

flow network G'

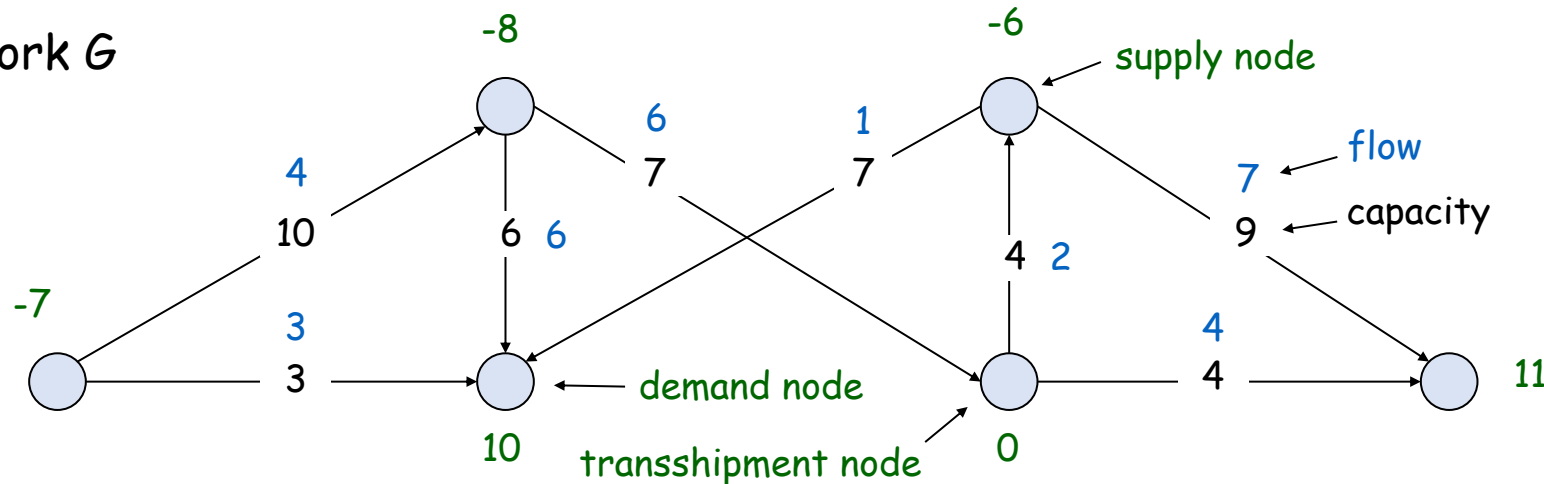




Circulation with Supplies and Demands

- **Def.** Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$ and node demands $d(v)$, a **circulation** is a function $f(e)$ that satisfies:
 - For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
 - For each $v \in V$: $\sum_{e \text{ into } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ [flow conservation]
- **Circulation problem.** Given (V, E, c, d) , find a circulation.

flow network G

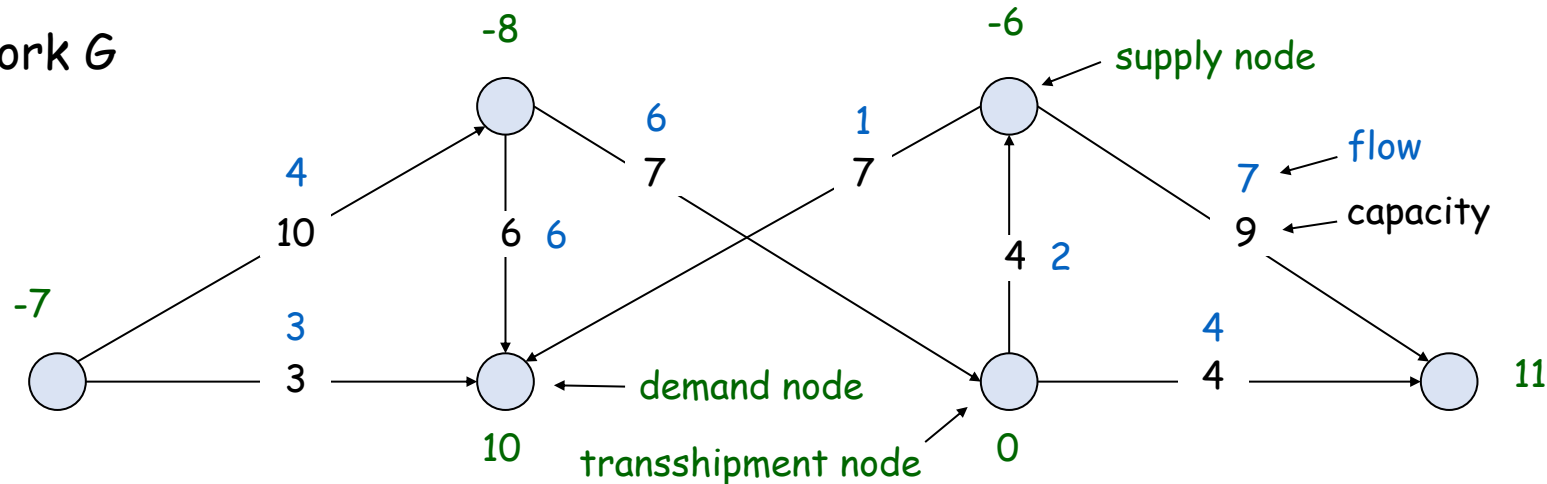




Circulation with Supplies and Demands

- **Def.** Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$ and node demands $d(v)$, a **circulation** is a function $f(e)$ that satisfies:
 - For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
 - For each $v \in V$: $\sum_{e \text{ into } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ [flow conservation]
- **Observation.** G has a circulation $\Rightarrow \sum_{v:d(v)>0} d(v) = \sum_{v:d(v)<0} -d(v)$.
total demand = total supply
- **Pf.** Sum of flow conservation for all nodes = 0.

flow network G

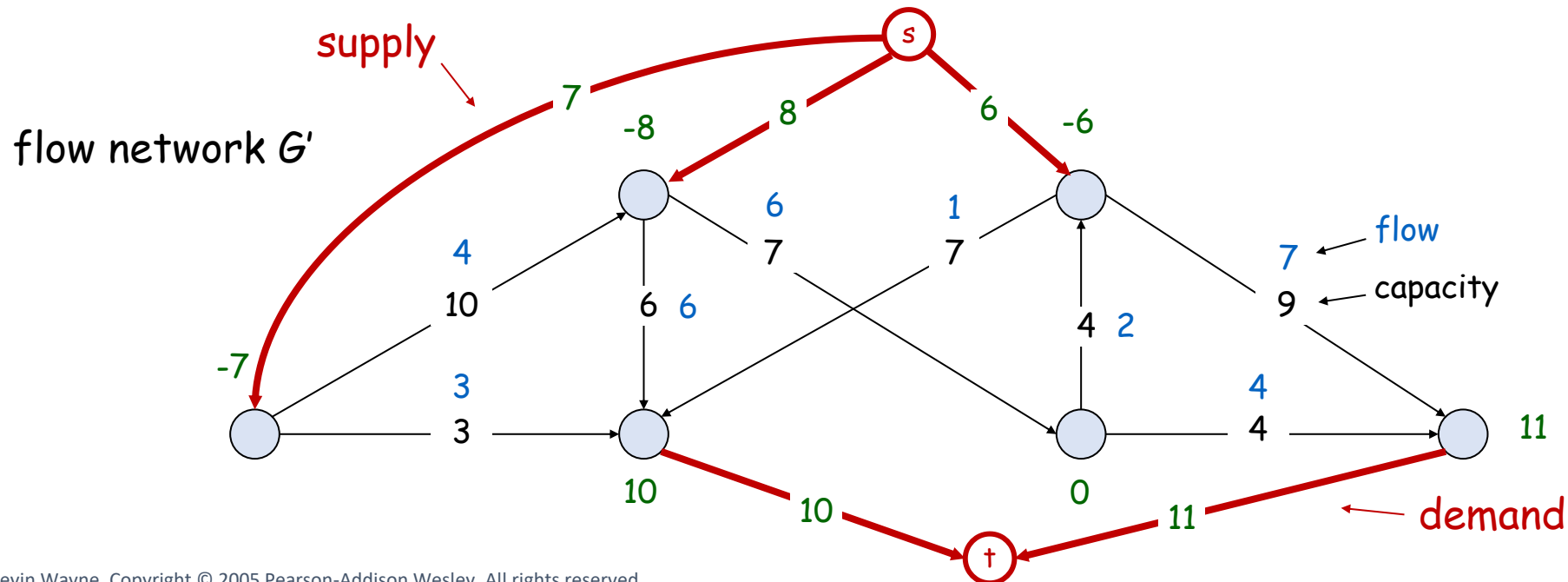




Circulation: Max-Flow Formulation

- **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 $d(v)$.



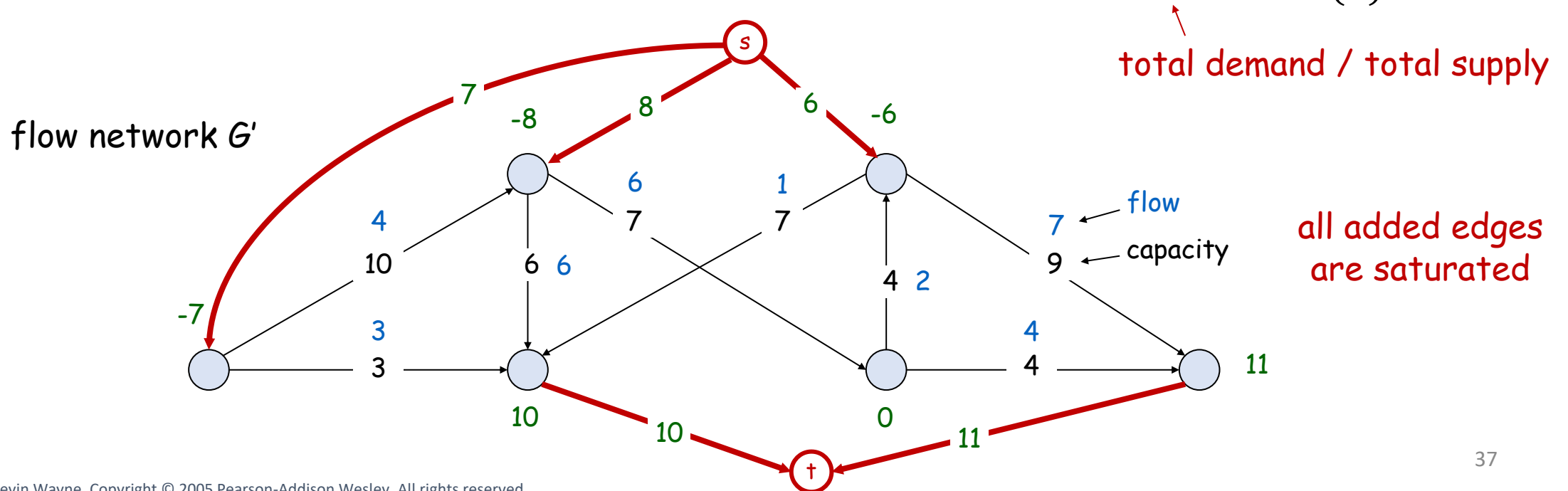


Circulation: Max-Flow Formulation

- **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 a circulation iff G' has max flow of value $D = \sum_{v:d(v)>0} d(v)$.





Circulation with Supplies and 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 + integrality theorem for max flow.
- **Theorem.** 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 sketch.** Look at min cut in G' .

exploit the relation between
cut in G' and node partition 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 Lower Bounds

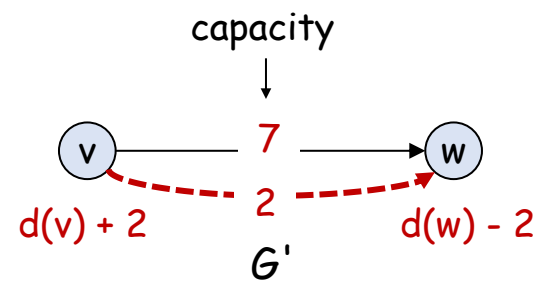
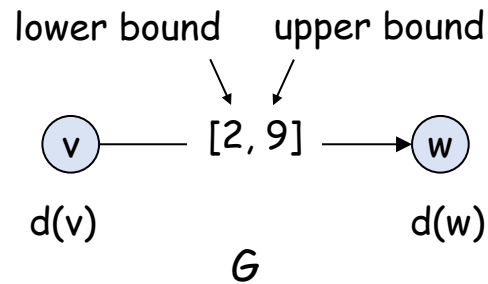
- **Def.** Given a digraph $G = (V, E)$ with edge capacities $c(e) \geq 0$, **lower bounds** $\ell(e) \geq 0$, and node demands $d(v)$, a **circulation** is a function $f(e)$ that satisfies:
 - For each $e \in E$: $\ell(e) \leq f(e) \leq c(e)$ [capacity]
 - For each $v \in V$: $\sum_{e \text{ into } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ [flow conservation]
- **Circulation problem with lower bounds.** Given (V, E, ℓ, c, d) , does there exist a feasible circulation?



Circulation with Lower Bounds

- **Max-flow formulation.**

- Model lower bounds as circulation with demands.
- Send $\ell(e)$ units of flow along edge e . ← this flow can then be abstracted away in G'



- **Theorem.** There exists a circulation in G iff there exists a circulation in G' . Moreover, if all demands, capacities, and lower bounds in G are integers, then there exists 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' .



10. Survey Design



Survey Design

- **Survey design.**

- Design survey asking n_1 consumers about n_2 products.
- Can survey consumer i about product j only 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 .

← one question per
consumer-product

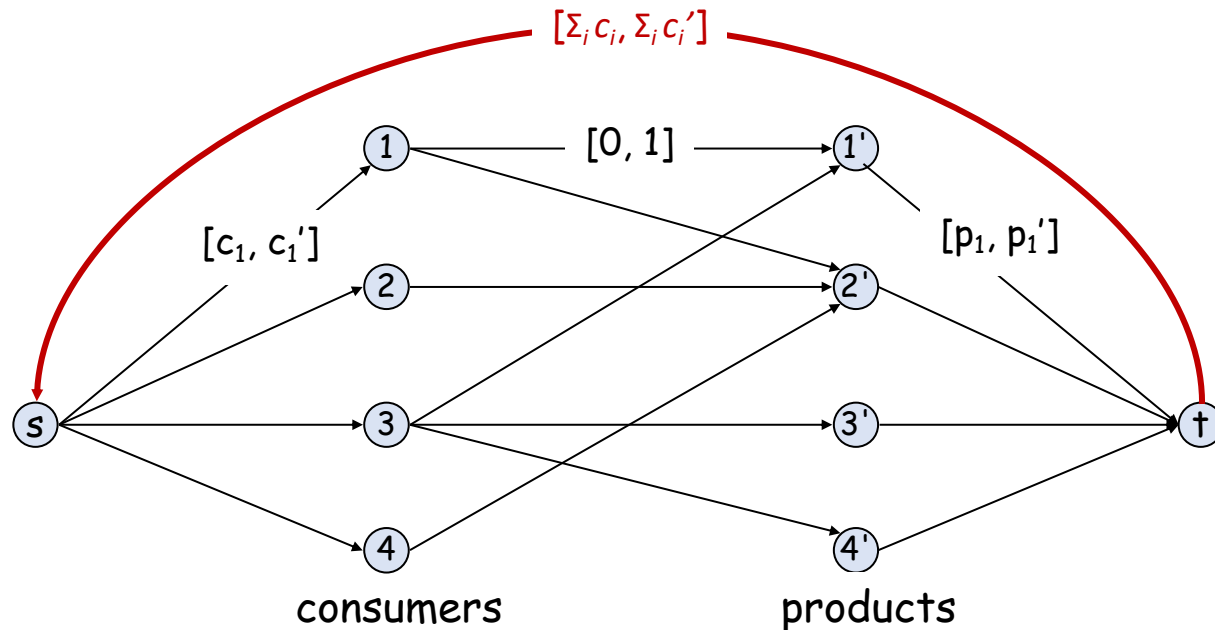
- **Goal.** Design a survey that meets these specs, if possible.

- **Bipartite perfect matching.** Special case when $c_i = c_i' = p_j = p_j' = 1$.



Survey Design: Circulation Formulation

- **Circulation formulation.** A circulation problem with lower bounds.
 - Add edge (i, j) if consumer i owns product j .
 - Add edge from s to consumer i . Add edge from product j to t .
 - Add edge from t to s with capacity $\sum_i c_i'$ and lower bound $\sum_i c_i$. All demands = 0.
- **Claim.** Integer circulation \Leftrightarrow feasible survey design.



← circulation can be solved by max-flow formulation



11. Airline Scheduling



Airline Scheduling

- **Airline scheduling.**

- Complex computational problem faced by airline carriers.
- Must produce schedules that are efficient in terms of equipment usage, crew allocation, and customer satisfaction.
- One of largest consumers of high-powered algorithmic techniques.

← even in presence of unpredictable events, such as weather and breakdowns

- **“Toy problem.”**

- Manage flight crews by reusing them over multiple flights.
- Input: set of k flights for a given day.
- Flight i leaves origin o_i at time s_i and arrives at destination d_i at time f_i .
- **Minimize** number of flight crews.

← one crew per flight

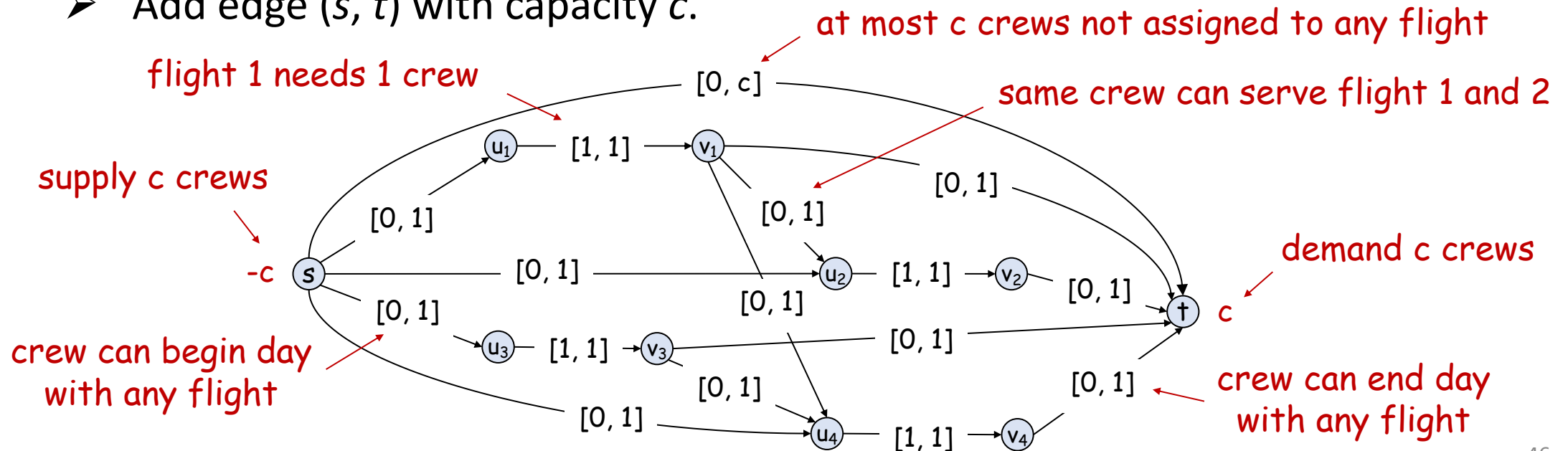


Airline Scheduling: Circulation Formulation

- **Circulation formulation.** (Check if c crews suffice.)

- For each i , add nodes u_i, v_i and edge (u_i, v_i) with lower bound and capacity 1.
- Add source s with demand $-c$, and edges (s, u_i) with capacity 1.
- Add sink t with demand c , and edges (v_i, t) with capacity 1.
- If flight j reachable from i , add edge (v_i, u_j) with capacity 1.
- Add edge (s, t) with capacity c .

u_i = start of flight i
 v_i = end of flight i





Airline Scheduling: Running Time

- **Theorem.** Airline scheduling problem can be solved in $O(k^3 \log k)$ time.
- **Pf.** k = number of flights.
 - $O(k)$ nodes, $O(k^2)$ edges.
 - c = number of crews (unknown).
 - At most k crews needed \Rightarrow solve $\log_2 k$ circulation problems.
 - Any flow value is between 0 and $k \Rightarrow \leq k$ augmentations per circulation problem.
 - Overall time = $O(k^3 \log k)$.
- **Note.** Can solve in $O(k^3)$ time by formulating as **minimum-flow problem**.

binary search for min value c^*





Airline Scheduling: Running Time

- **Remark.** We solved a toy version of a real problem.
- **Real-world problem models countless other factors:**
 - Flight crews can fly only a certain number of hours in a given time window.
 - Need optimal schedule over planning horizon, not just one day.
 - Flights don't always leave or arrive on schedule.
 - Simultaneously optimize both flight schedule and fare structure.
- **Message.**
 - Our solution is a generally useful technique for efficient reuse of limited resources but trivializes real airline scheduling problem.
 - Flow techniques useful for solving airline scheduling problems (and are widely used in practice).
 - Running an airline efficiently is a very difficult problem.



12. Image Segmentation



Image Segmentation

- **Image segmentation.**
 - Divide image into coherent regions.
 - Central problem in image processing.
- **Ex.** Separate human and robot from background scene.

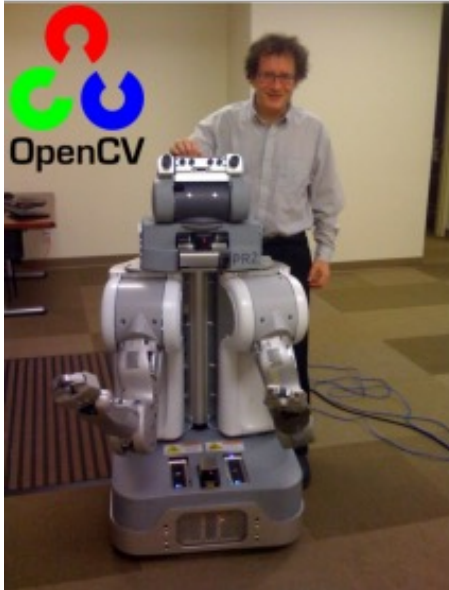
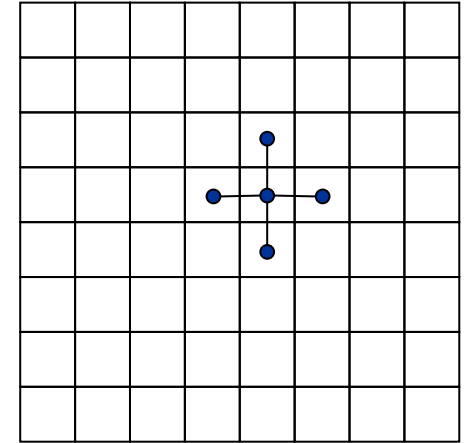




Image Segmentation

- **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.
- $p_{ij} \geq 0$ for each $(i, j) \in E$ 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:
$$q(A, B) = \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

\nearrow
foreground

\nearrow
background



Image Segmentation

- **Formulate as min-cut problem.**

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

- **Turn into minimization problem.**

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

is equivalent to **minimizing** $\left(\sum_{i \in A \cup B} a_i + \sum_{j \in A \cup B} b_j \right) - q(A, B) = \sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}|=1}} p_{ij}$

↑
constant

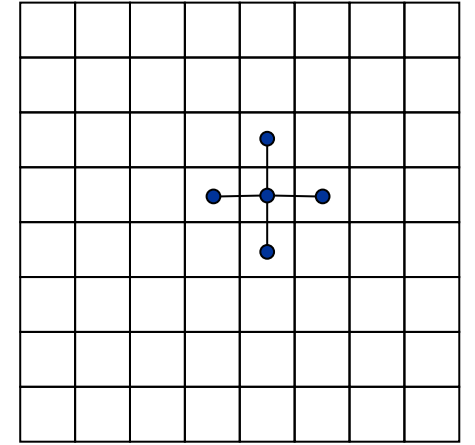




Image Segmentation

- **Formulate as min-cut problem $G' = (V', E')$.**

- Include node for each pixel.
- Use **two anti-parallel edges** instead of undirected edge.
- Add source s to correspond to foreground.
- Add sink t to correspond to background.

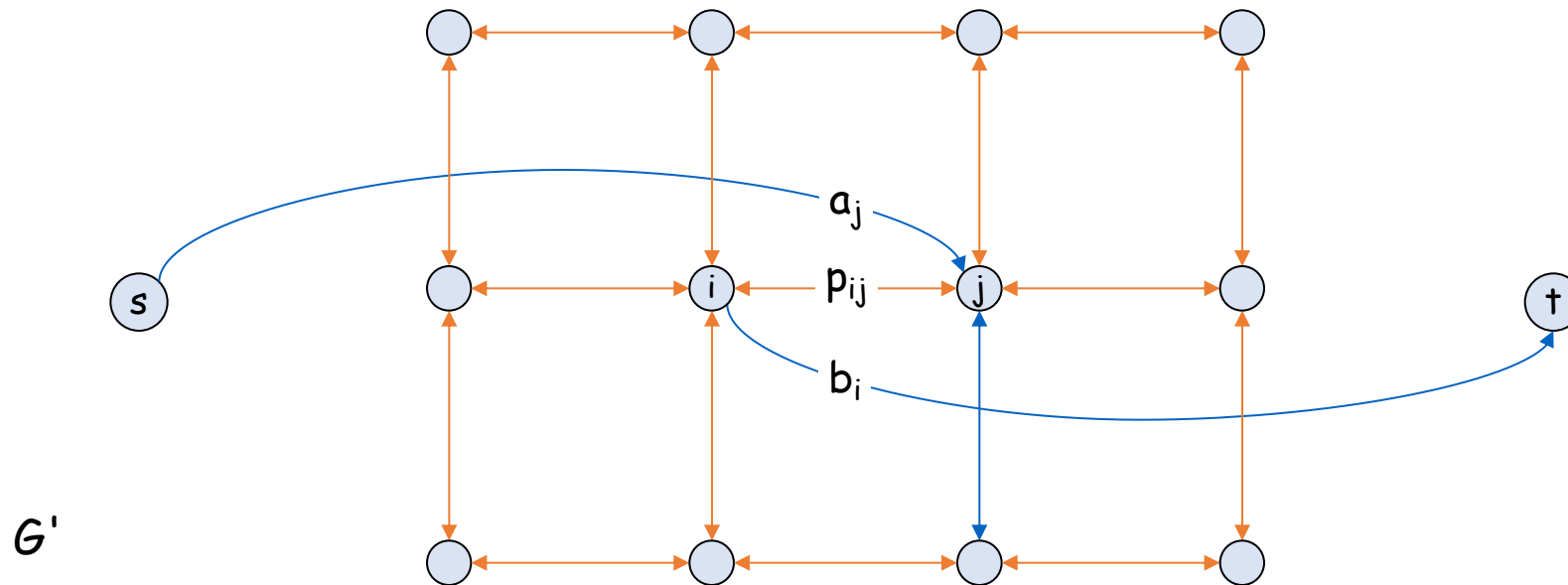
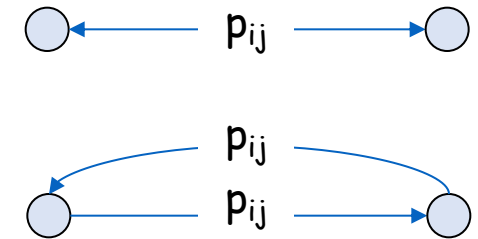
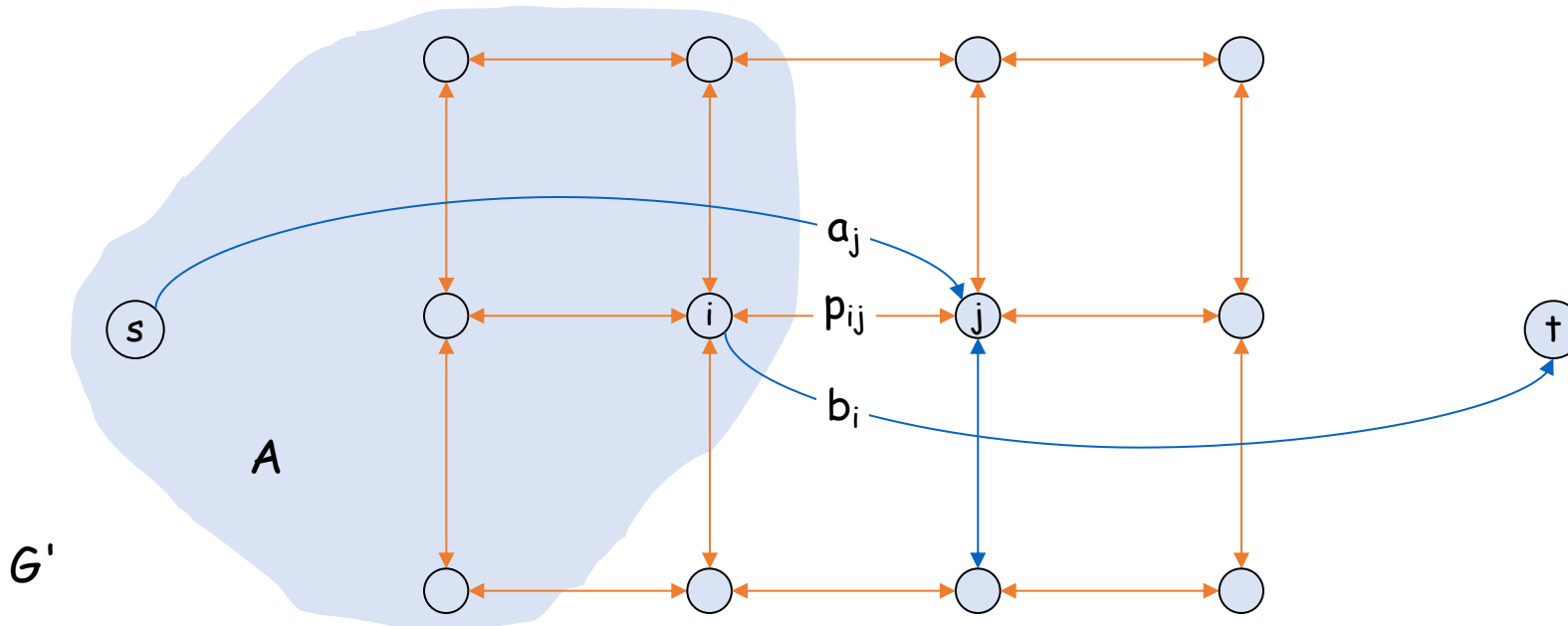




Image Segmentation

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

- A = foreground. $cap(A, B) = \sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E' \\ i \in A, j \in B}} p_{ij}$
 - Precisely the quantity we want to minimize.
- if i and j on different sides, p_{ij} counted exactly once





More on Network Flow Problems

- **More applications:**

- Project selection (maximum weight closure problem). [Textbook, Section 7.11]
- Baseball elimination. [Textbook, Section 7.12]

- **Problems solved by more advanced network flow algorithms:**

- Minimum-cost perfect matching. [Textbook, Section 7.13]
- Minimum-cost flow (as a general problem).