CSE 6140/ CX 4140:

Computational Science and Engineering ALGORITHMS

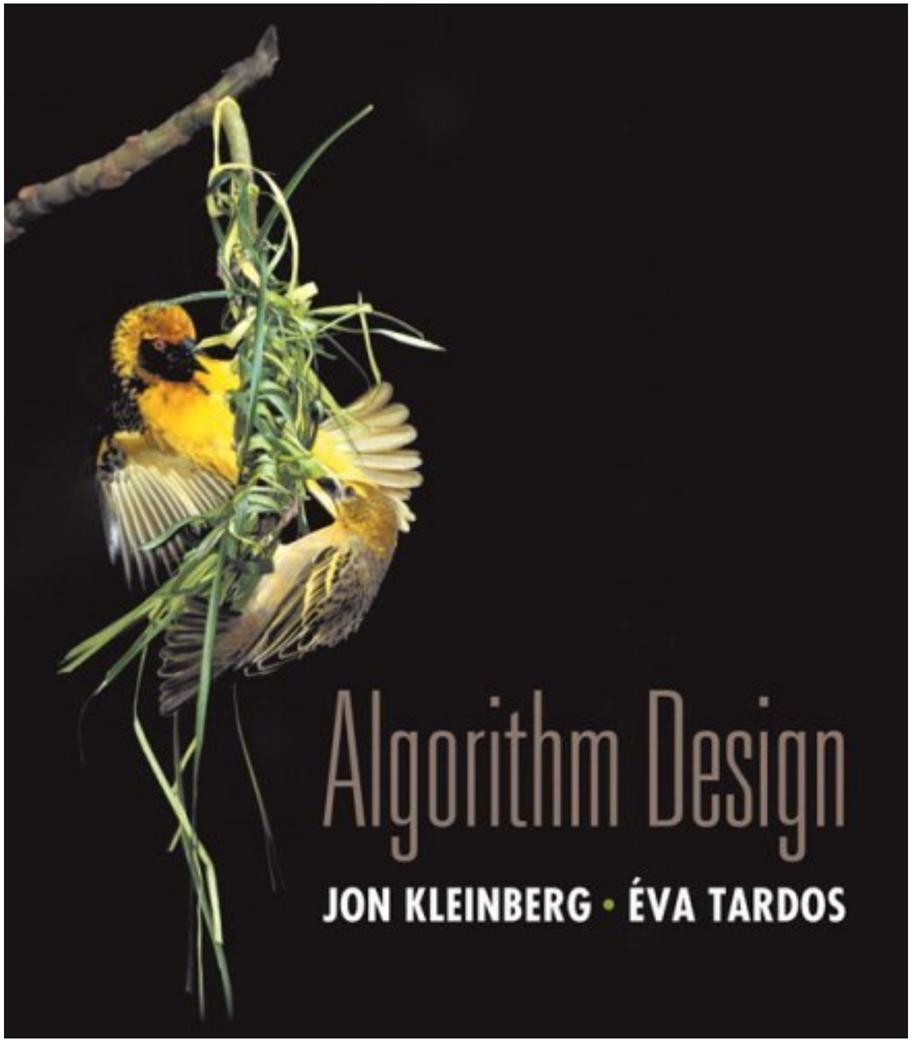
Instructor: Anne Benoit

Visiting Associate Professor, CSE

Based on slides by Bistra Dilkina

CLRS: Chapter 26 & KT: Chapter 7

Network flows – Part 3



Slides by Kevin Wayne.

KT 7.6

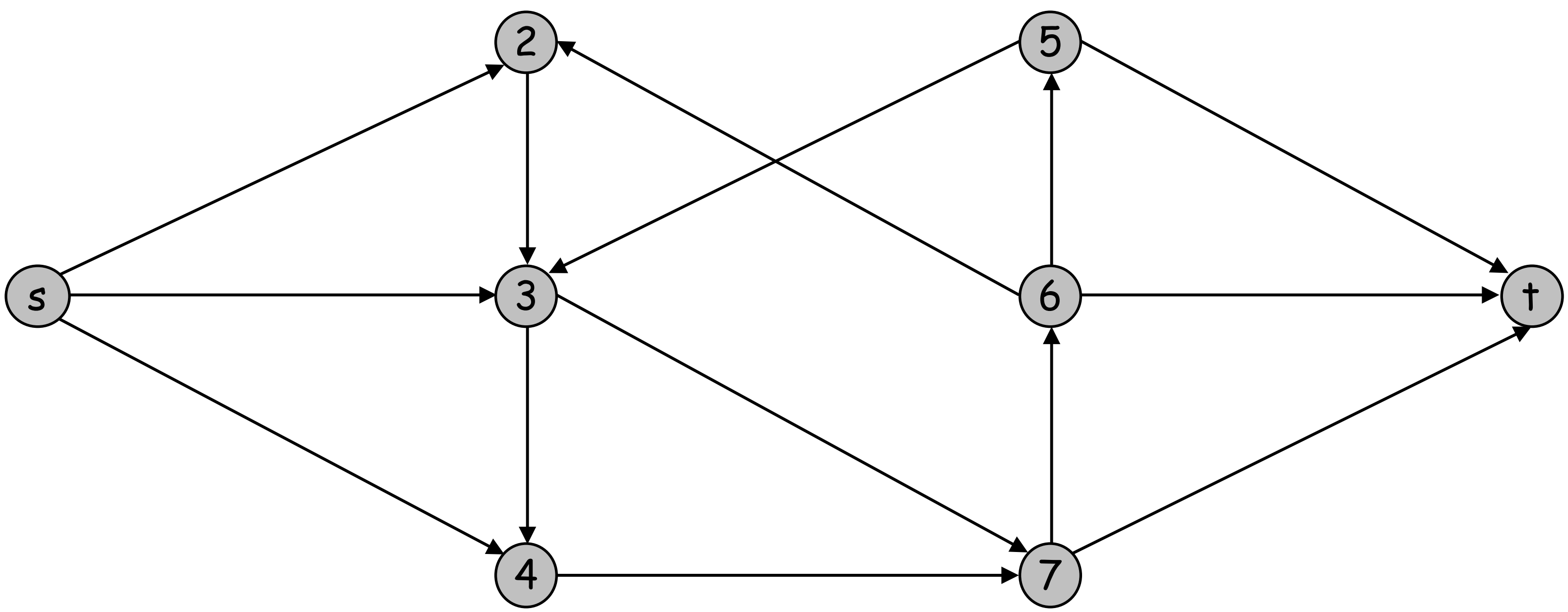
Disjoint Paths

Edge Disjoint Paths

Disjoint path problem. Given a directed graph $G = (V, E)$ 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 (Resilience of networks).

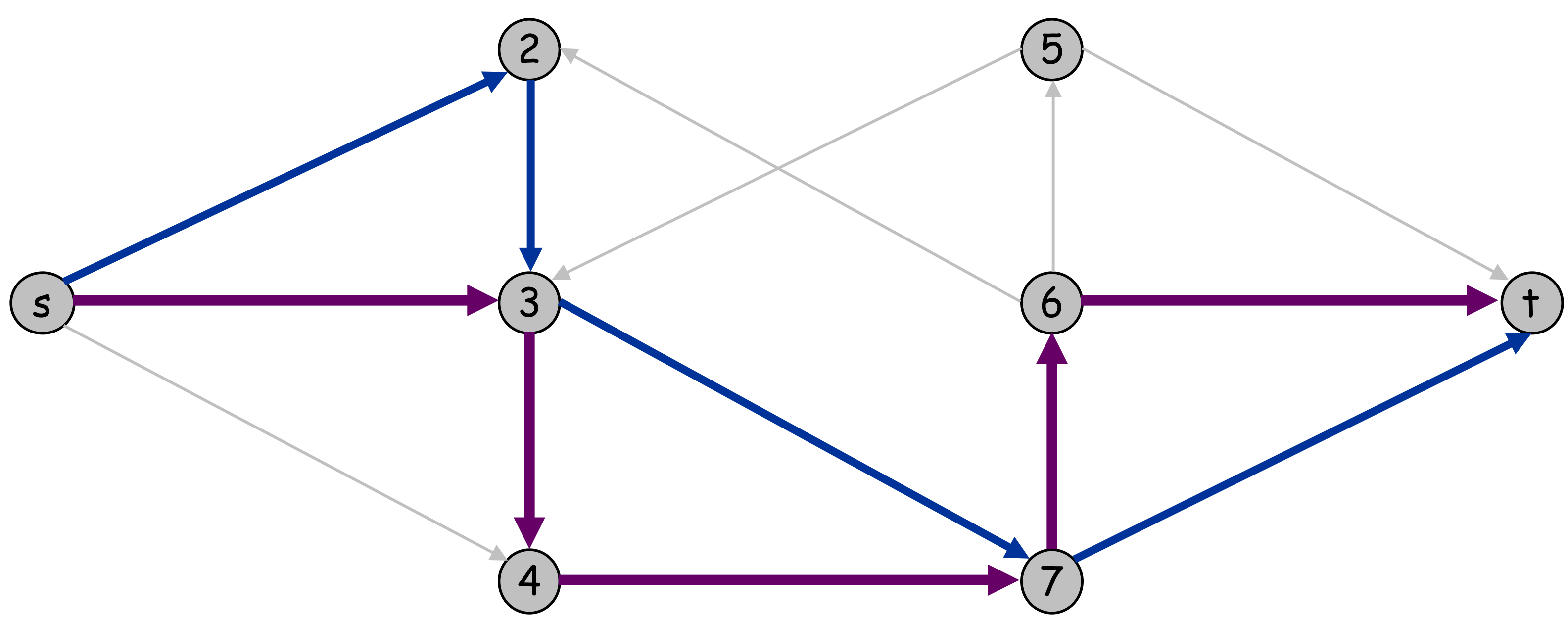


Edge Disjoint Paths

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

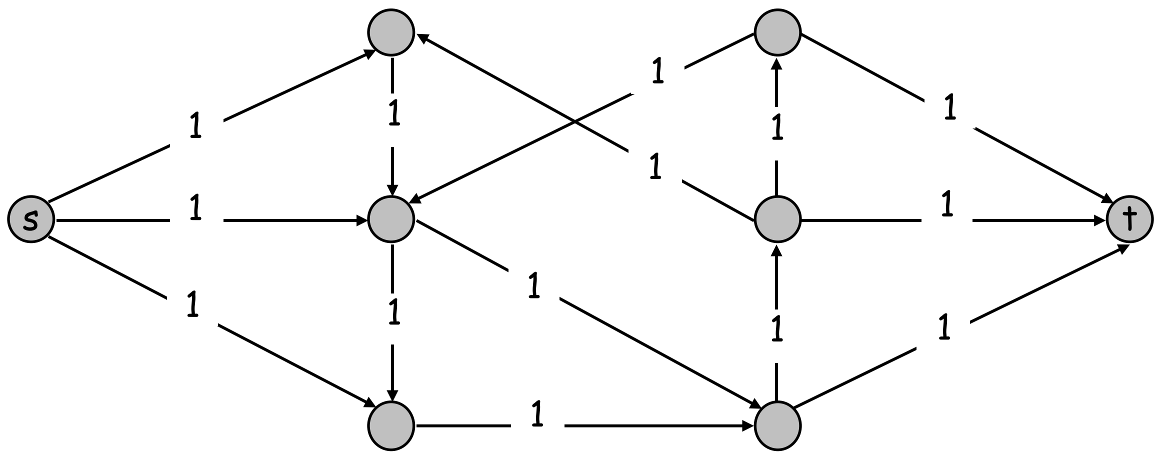
Def. Two paths are edge-disjoint if they have no edge in common.

Ex: Communication networks (Resilience of networks).



Edge Disjoint Paths

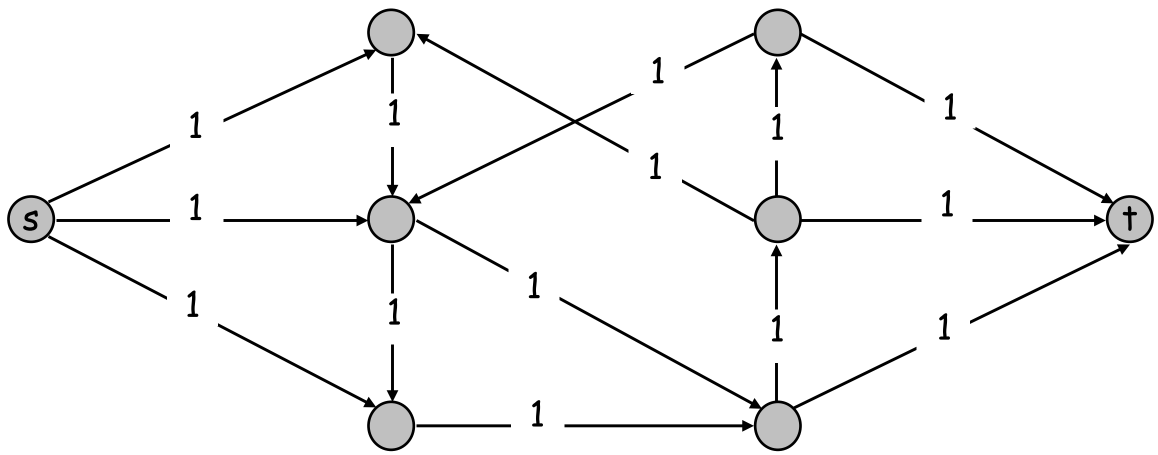
Max flow formulation: assign unit capacity to every edge.



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

Edge Disjoint Paths

Max flow formulation: assign unit capacity to every edge.



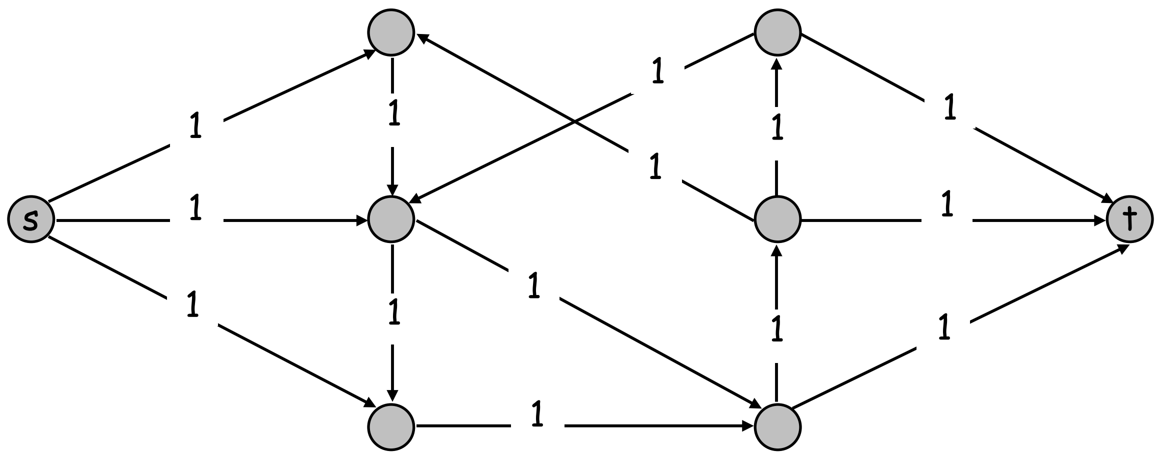
Theorem. Max number edge-disjoint s-t paths equals max flow value.

Pf. Max edge disjoint paths ≤ maxflow

- Suppose there are k edge-disjoint paths $P_1, \ldots, P_k$.
- Set $f(e) = 1$ if e participates in some path P_i ; else set $f(e) = 0$.
- Conservation of flow is preserved because we are adding 1 unit of flow along each of the given paths from s to t.
- Since paths are edge-disjoint, capacities are respected and f is a flow of value k (leaving s).

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. Max edge disjoint paths ≥ maxflow

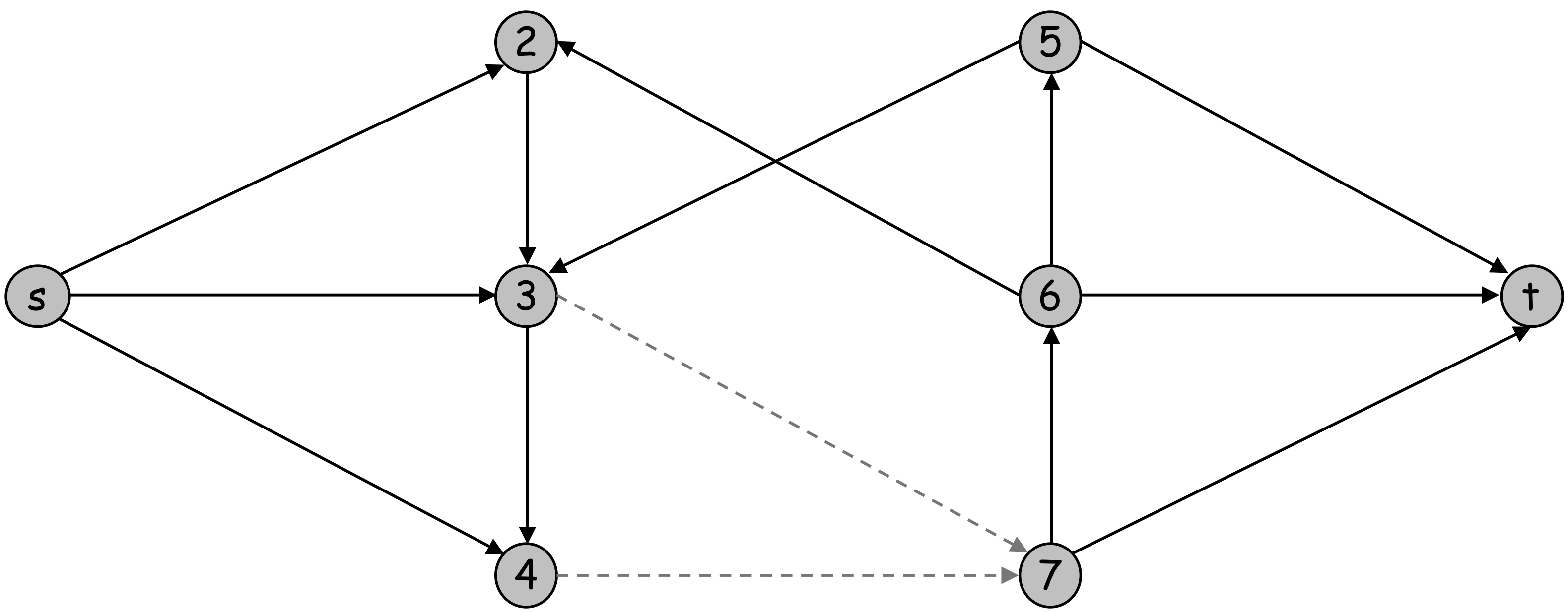
- Suppose max flow value is k.
- Integrality theorem $\Rightarrow$ there exists 0-1 flow f of value k.
- Consider edge (s, u) with f(s, u) = 1.
  - by conservation, there exists an edge (u, v) with f(u, v) = 1
  - continue until reach t, always choosing a new edge with flow (Path)
- There has to be k edges with flow out of s (hence can follow k Paths).
- Produces k (not necessarily simple) edge-disjoint paths, since c(e)=1.

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 ⊆ E disconnects t from s if all s-t paths uses at least one edge in F.
(That is, removing F would make t unreachable from s.)

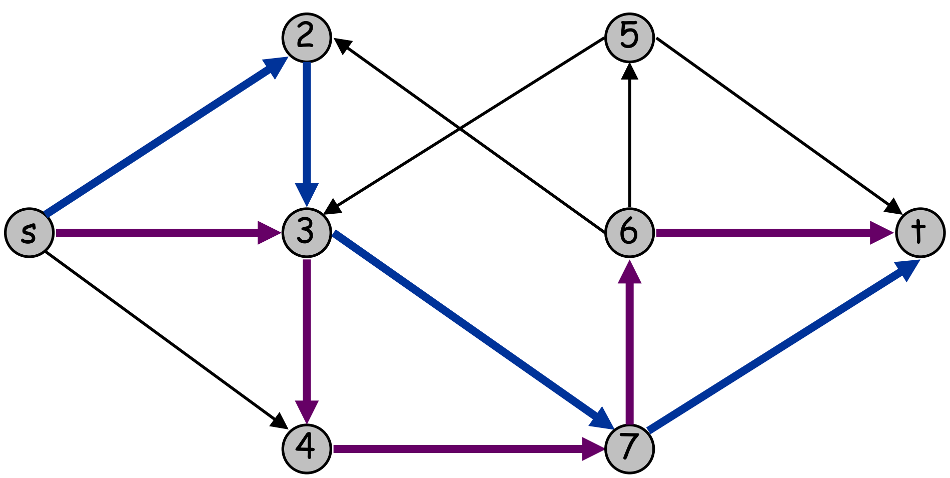


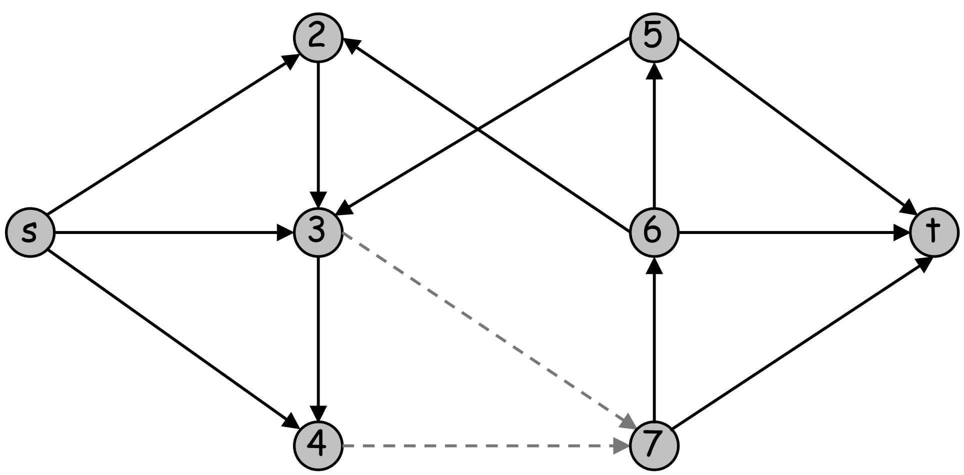
Edge Disjoint Paths and Network Connectivity

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. Max num edge-disjoint paths ≤ min number of edges to remove

- 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, and edge-disjoint paths cannot share edges
- Hence, the number of edge-disjoint paths is at most k.



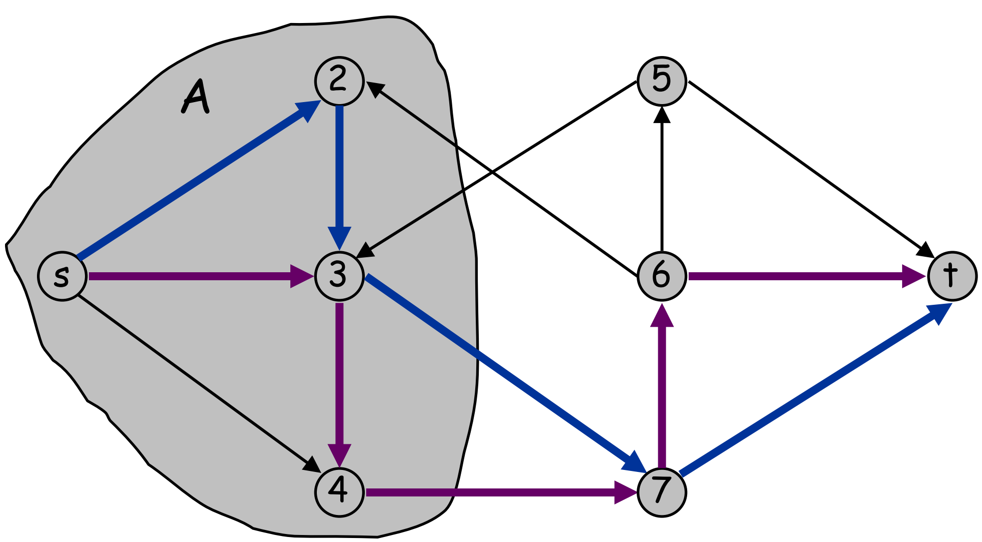


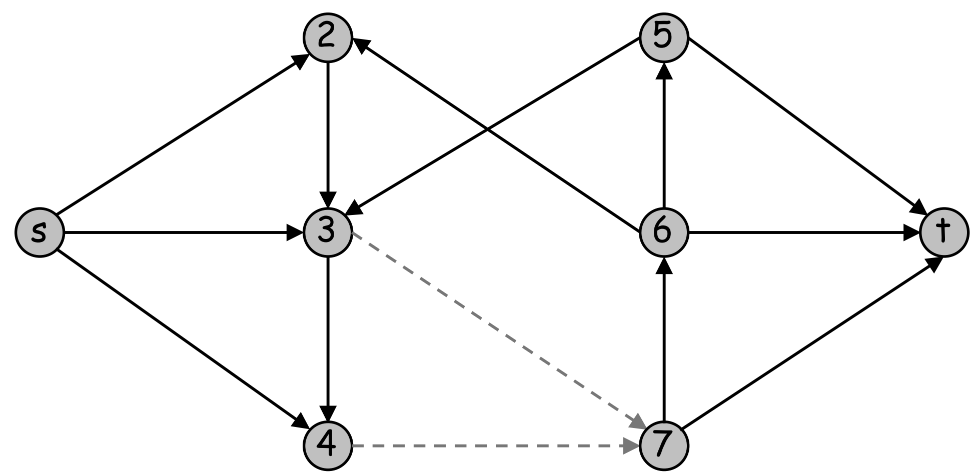
Edge Disjoint Paths and Network Connectivity

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. Max num edge-disjoint paths ≥ min number of edges to remove

- Suppose max number of edge-disjoint paths is k.
- Then max flow value is k (from before).
- Max-flow min-cut $\Rightarrow$ exists cut (A, B) of capacity k.
- Let F be set of edges going from A to B, each has capacity of 1.
- |F| = k and disconnects t from s.





KT 7.3

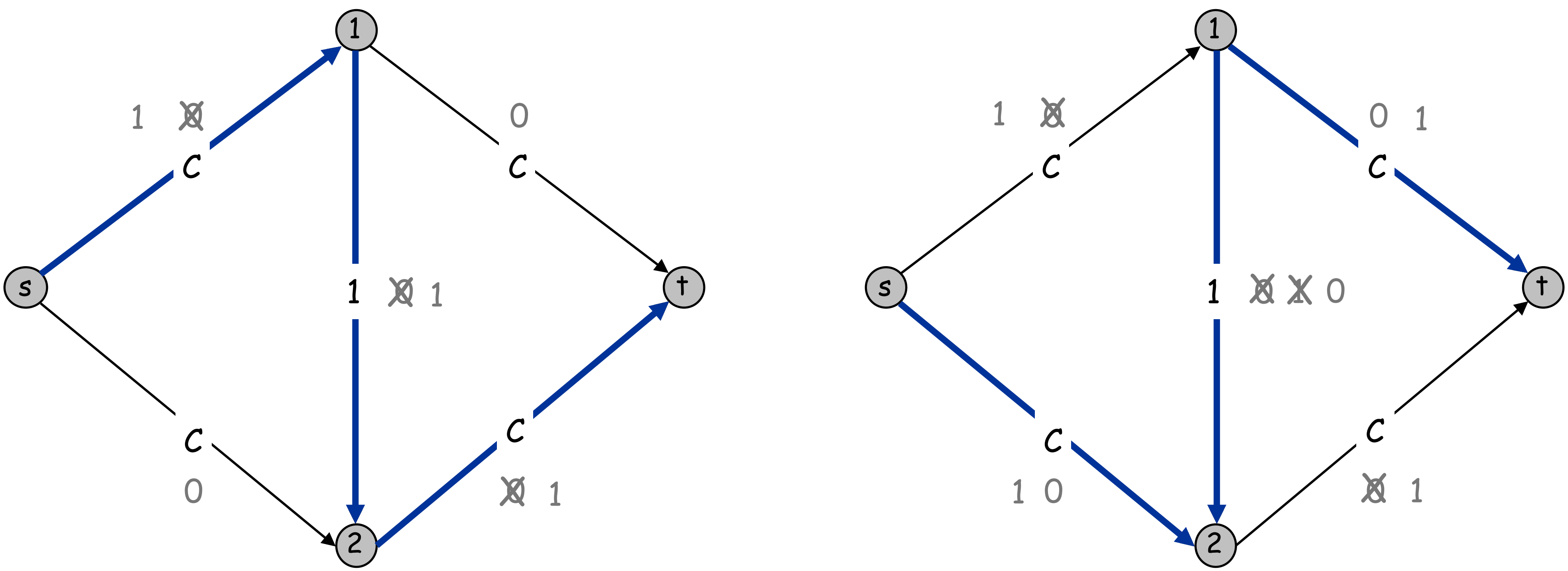
Faster algorithms for max flow

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 nC iterations.



Intuition: we are choosing the wrong paths!

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms (in log C).
- 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.

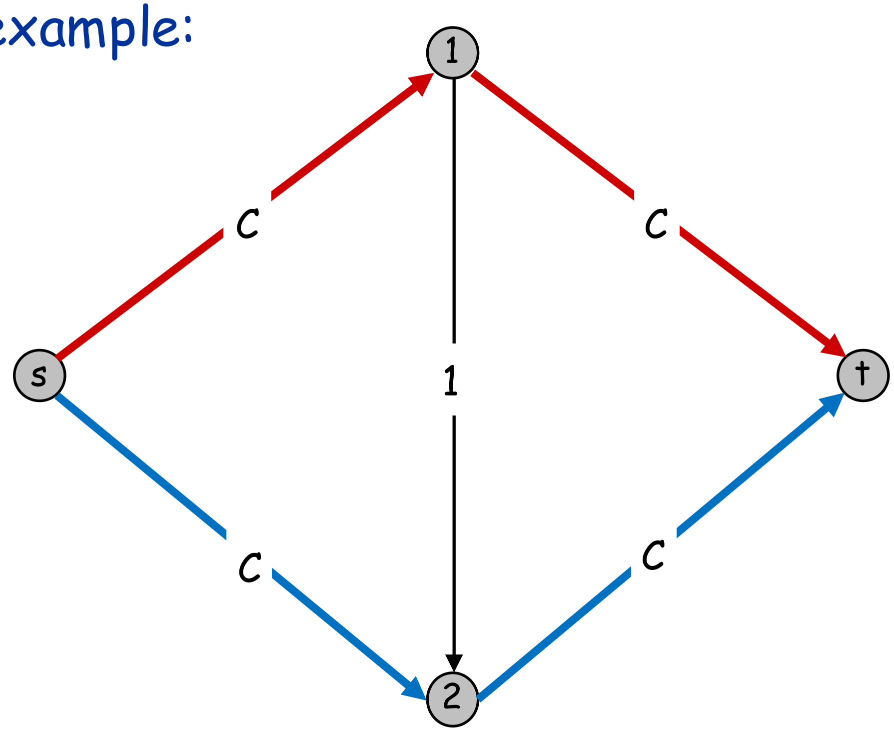
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

- Max bottleneck capacity. (Fat)
- Sufficiently large bottleneck capacity. (Scaling)
- Fewest number of edges. (Shortest)

Edmonds-Karp Algorithm

Ford-Fulkerson with shortest paths (in terms of number of hops).

Previous example:



Edmonds-Karp finds s-1-t and s-2-t ☺

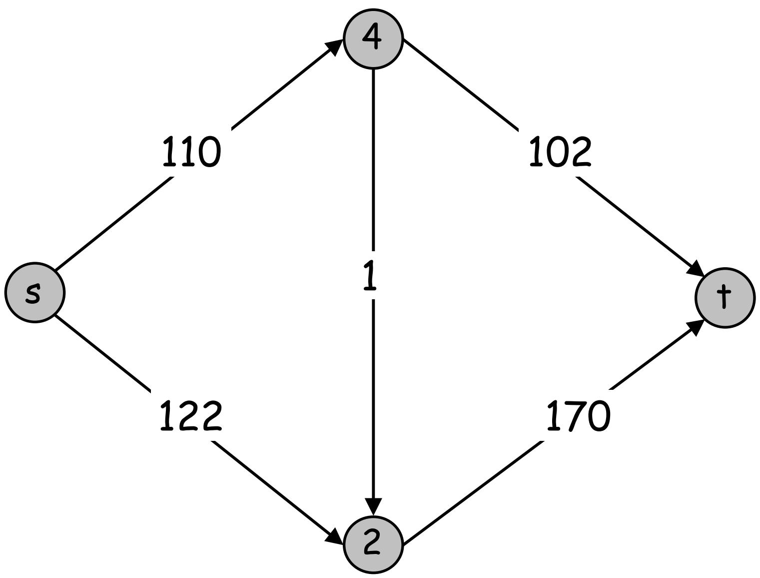
Theorem: Edmonds-Karp makes at most $O(nm)$ flow augmentations, hence a complexity in $O(n\,m^2)$, with capacities in $\mathbb{R}^+$.

Capacity Scaling

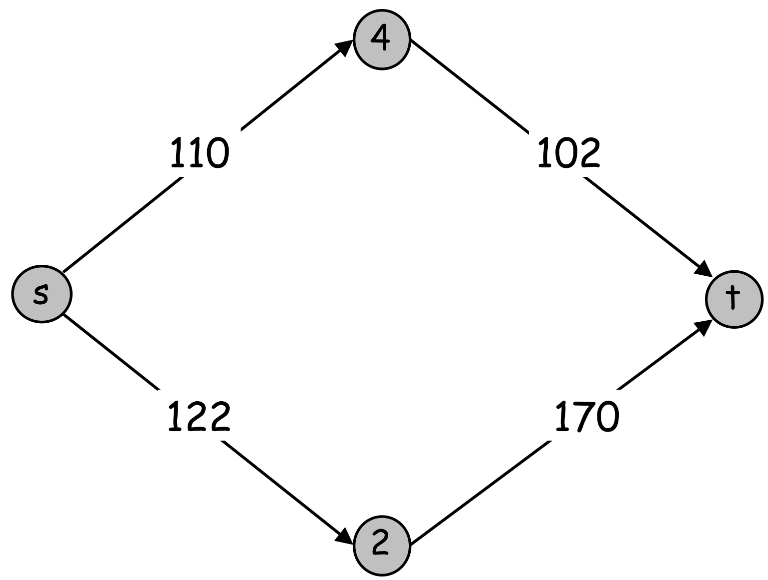
Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

Back to integer capacities.

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



Residual graph G_f



$G_f(100)$

Capacity Scaling Algorithm

```
Scaling-Max-Flow(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Δ ← smallest power of 2 greater than or equal to C
   Gf ← residual graph

   while (Δ ≥ 1) {
      Gf(Δ) ← Δ-residual graph
      while (there exists augmenting path P in Gf(Δ)) {
         f ← augment(f, c, P) // augment flow by ≥ Δ
         update Gf(Δ)
      }
      Δ ← Δ / 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.
(still holds)

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 $C \le \Delta < 2C$. 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 f^* is at most $v(f) + m\,\Delta$. ← proof on next slide

(Sanity check: $|v(f^*) - v(f)| \le m\Delta$, and Δ shrinks, so $v(f)$ converges towards $v(f^*)$)

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

- Let f be the flow at the end of the previous scaling phase.
- Lemma 2 $\Rightarrow$ $v(f^*) \le v(f) + m\,(2\Delta)$.
- 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

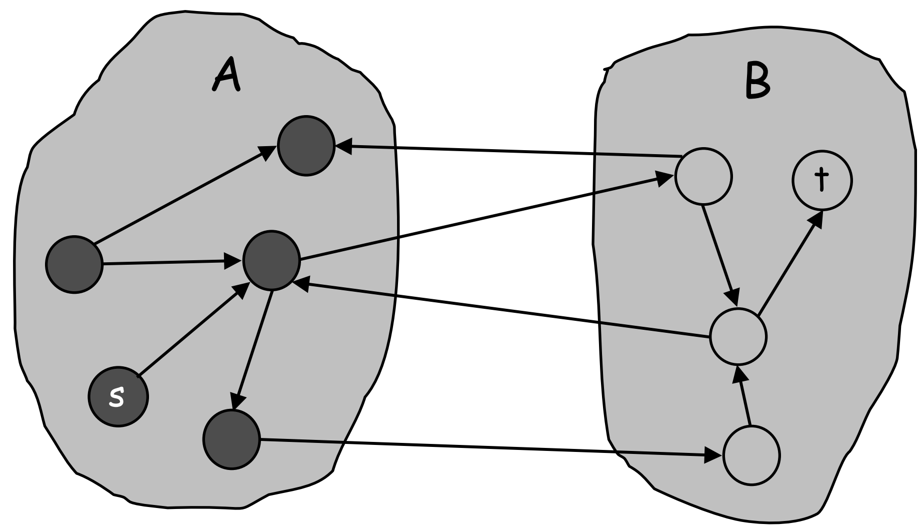
Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $v(f) + m\,\Delta$.

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 $cap(A, B) \leq v(f) + m\,\Delta$.
- 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, $t \notin A$.

$c(e)-f(e) < \Delta$ $\qquad$ $f(e) < \Delta$

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



original network

So $cap(A,B) - v(f) \leq m\Delta$

$\Rightarrow v(f^*) - v(f) \leq cap(A,B) - v(f) \leq m\Delta$

Best Known Algorithms For Max Flow

Reminder: The scaling max-flow algorithm runs in $O(m^2 \log C)$ time.

Compare to:

- $O(n\, m\, C)$ (FF method)
- $O(n\, m^2)$ (Edmonds-Karp)

Currently there are other algorithms that run in time

- $O(mn \log n)$
- $O(n^3)$
- $O(\min(n^{2/3}, m^{1/2})\, m \log n \log C)$

Active topic of research:

- Flow algorithms for specific types of graphs
- Special cases (bipartite matching, etc)
- Multi-commodity flow
- ...

7.10 Image Segmentation

Image Segmentation

Image segmentation.

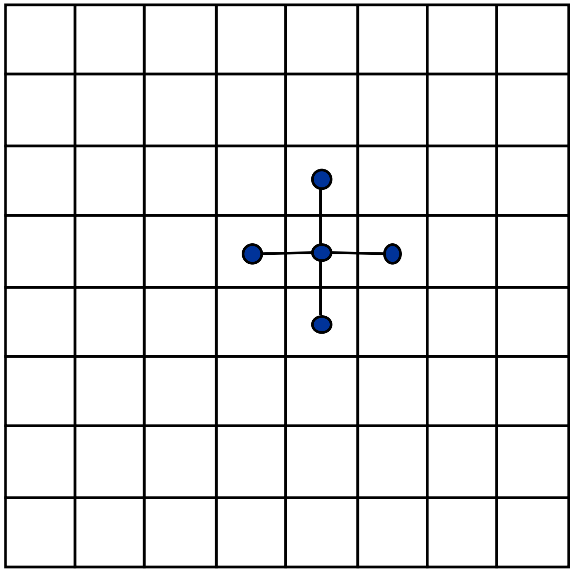
- Central problem in image processing.
- Divide image into coherent regions.

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

$\underset{\text{foreground}}{A}$, $\underset{\text{background}}{B}$

Image Segmentation

A problem that also partitions the nodes of a graph is **Min Cut**.
Formulate as min cut problem. But our Image Segmentation problem is:

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

Turn into minimization problem.

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

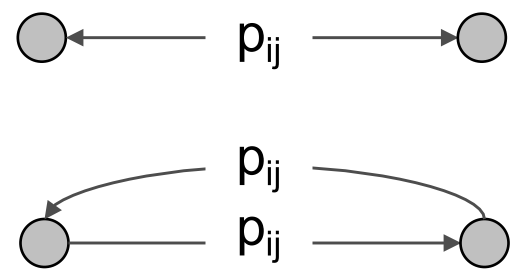
  is equivalent to minimizing $$\underbrace{\left(\sum_{i \in V} a_i + \sum_{j \in V} b_j\right)}_{\text{a constant}} - \sum_{i \in A} a_i - \sum_{j \in B} b_j + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

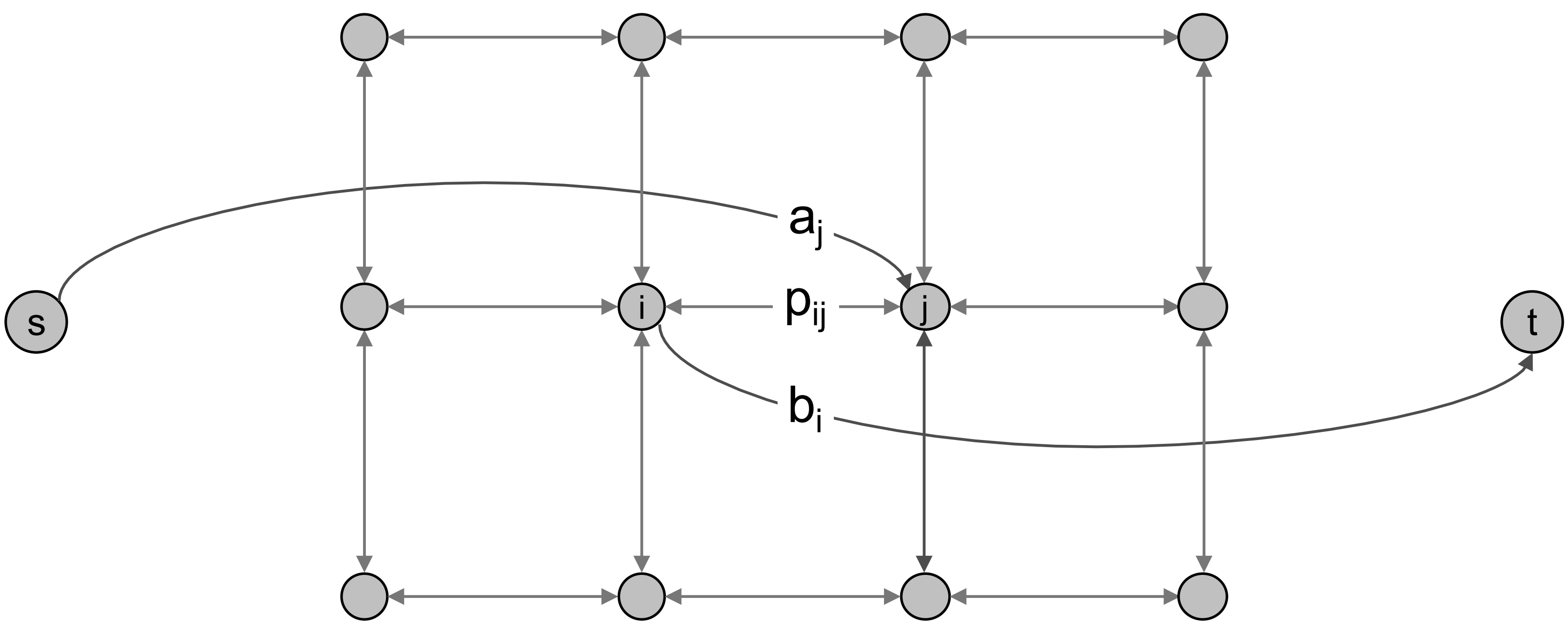
- or alternatively $$\sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{\substack{(i,j) \in E \\ |A \cap \{i,j\}| = 1}} p_{ij}$$

Image Segmentation

Formulate as min cut problem (also looks at all 2 partitions of nodes).

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





G'

Image Segmentation

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

- A = foreground.

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

$\longleftarrow$ if i and j on different sides, p_{ij} counted exactly once

- Precisely the quantity we want to minimize in MinCut.