CSE 421

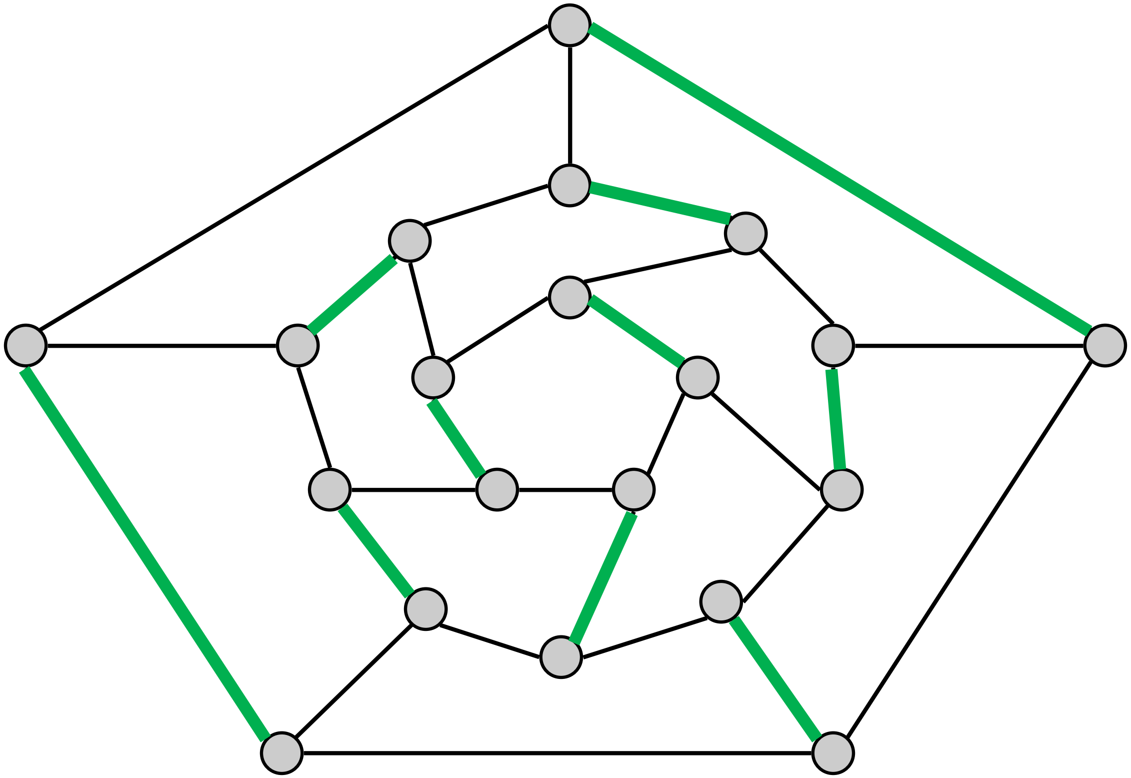
Application of Max Flow

Yin Tat Lee

Applications of Max Flow:
Bipartite Matching

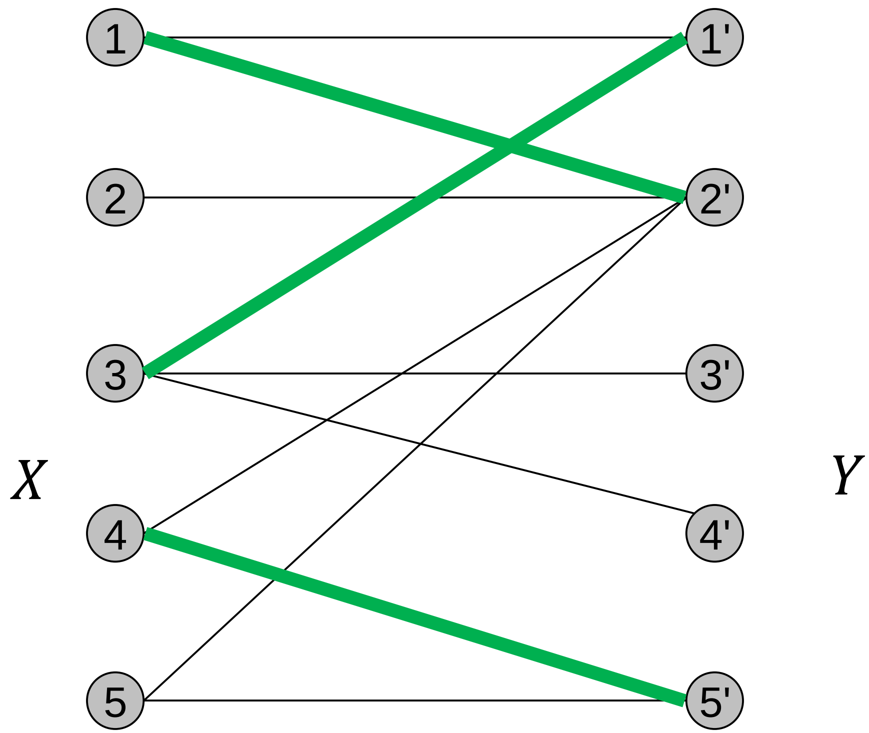
Maximum Matching Problem

Given an undirected graph $G = (V, E)$.
A set $M \subseteq E$ is a matching if each vertex appears in at most 1 edge in M.
Goal: find a matching with largest cardinality.



Bipartite Matching Problem

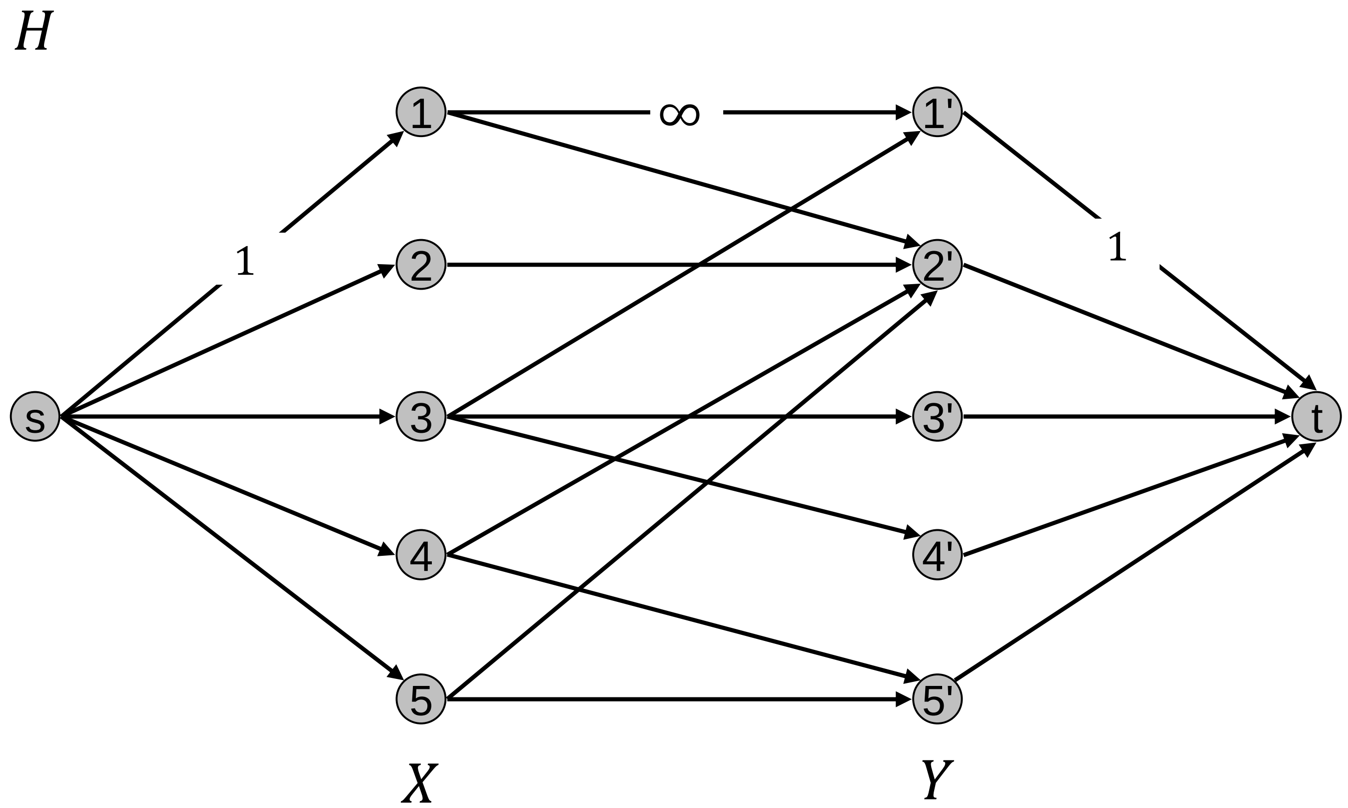
Given an undirected bipartite graph $G = (X \cup Y, E)$
A set $M \subseteq E$ is a matching if each vertex appears in at most 1 edge in M.
Goal: find a matching with largest cardinality.



Bipartite Matching using Max Flow

Create digraph H as follows:

- Orient all edges from X to Y and assign infinite (or unit) capacity.
- Add source s, and unit capacity edges from s to each node in X.
- Add sink t, and unit capacity edges from each node in Y to t.



Bipartite Matching: Proof of Correctness

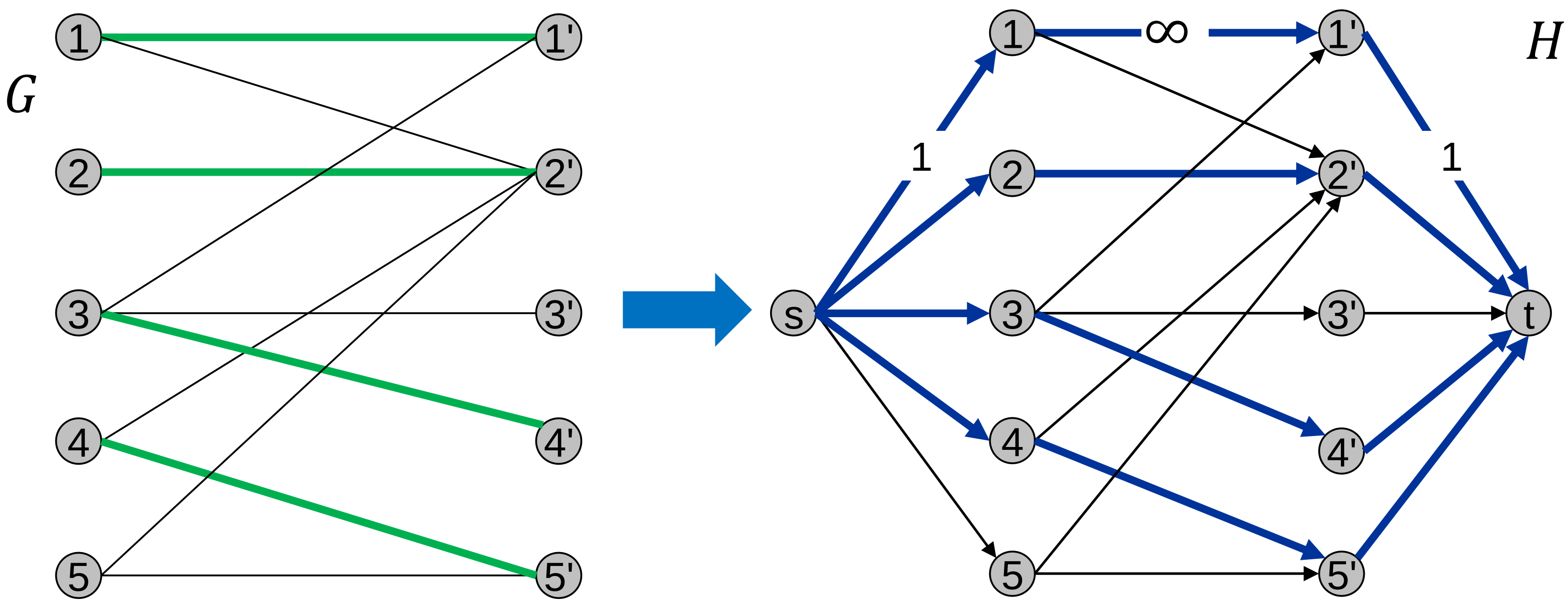
Thm. Max cardinality matching in G = value of max flow in H.

Pf. (matching val ≤ maxflow val)

Given max matching M of cardinality k.

Consider flow f that sends 1 unit along each of k edges of M.

f is a flow and has cardinality k.



Bipartite Matching: Proof of Correctness

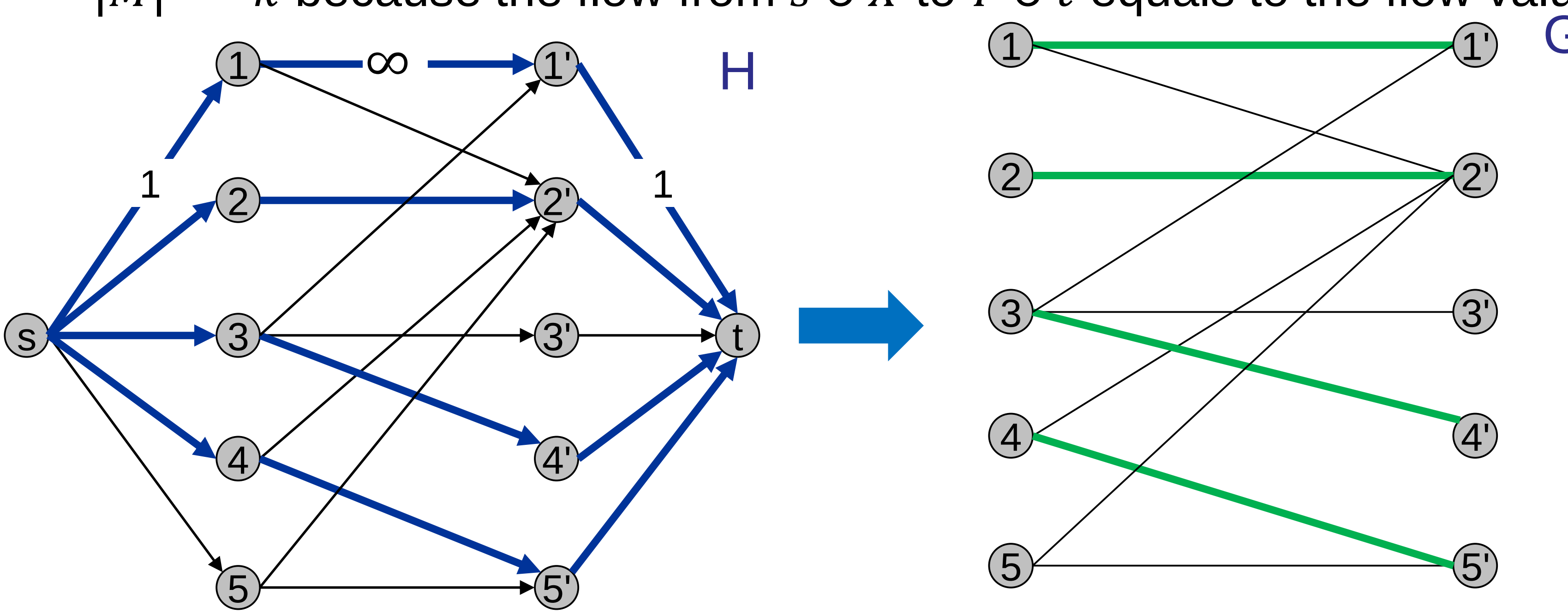
Thm. Max cardinality matching in G = value of max flow in H.

Pf. (of matching val ≥ flow val) Let f be a max flow in H of value k.

Integrality theorem $\Rightarrow$ k is integral and we can assume f is 0-1.

Consider M = set of edges from X to Y with $f(e) = 1$.

- each node in X and Y participates in at most one edge in M
- $|M| = k$ because the flow from $s \cup X$ to $Y \cup t$ equals to the flow value k.



Applications of Max Flow:
Perfect Bipartite Matching

Perfect Bipartite Matching

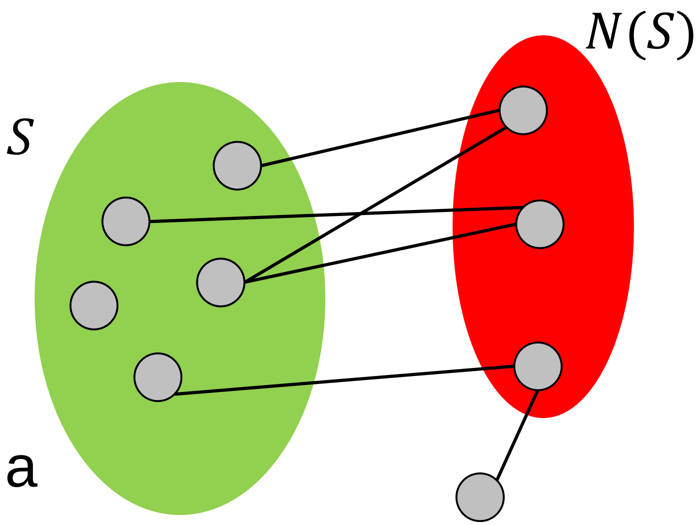
Def. A matching $M \subset E$ is perfect if each node appears in exactly one edge in M.

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

Structure of bipartite graphs with perfect matchings:

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

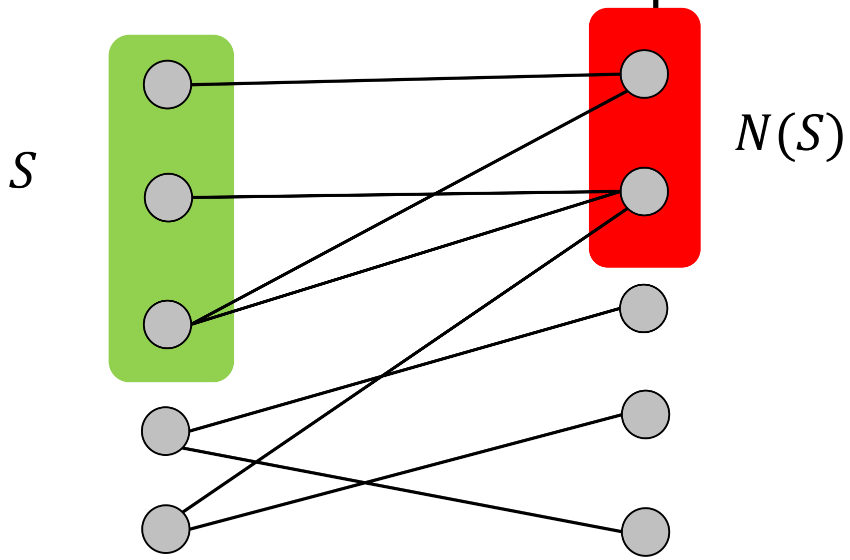
Perfect Bipartite Matching: N(S)



Def. 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 has a perfect matching, then $|N(S)| \geq |S|$ for all subsets $S \subset X$.

Pf. Each $v \in S$ has to be matched to a unique node in $N(S)$.



Marriage Theorem

Thm: [Frobenius 1917, Hall 1935] Let $G = (X \cup Y, E)$ be a bipartite graph with $|X| = |Y|$.

Then, G has a perfect matching iff $|N(S)| \geq |S|$ for all subsets $S \subseteq X$.

Pf. $\Rightarrow$

This was the previous observation.

If $|N(S)| < |S|$ for some S, then there is no perfect matching.

Marriage Theorem

Pf. $\exists S \subseteq X$ s.t., $|N(S)| < |S| \Leftarrow$ G does not a perfect matching

Formulate as a max-flow and let (A, B) be the min s-t cut

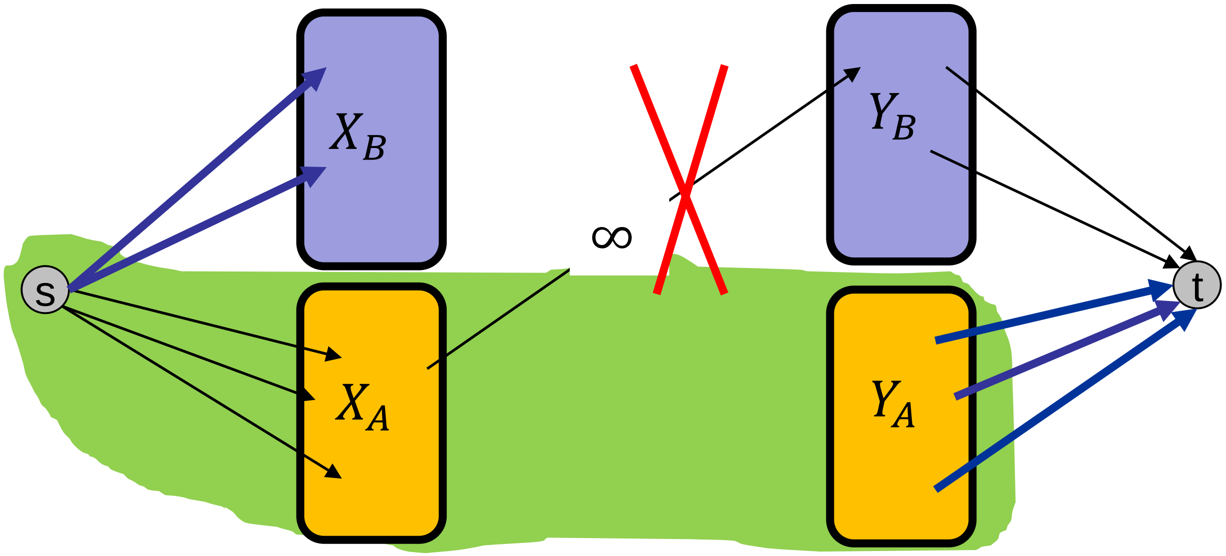
G has no perfect matching => $v(f^*) < |X|$. So, $cap(A, B) < |X|$

Define $X_A = X \cap A, X_B = X \cap B, Y_A = Y \cap A$

Then, $cap(A, B) \geq |X_B| + |Y_A|$

Since min-cut does not use ∞ edges, $N(X_A) \subseteq Y_A$

$|N(X_A)| \leq |Y_A| \leq cap(A, B) - |X_B| = cap(A, B) - |X| + |X_A| < |X_A|$



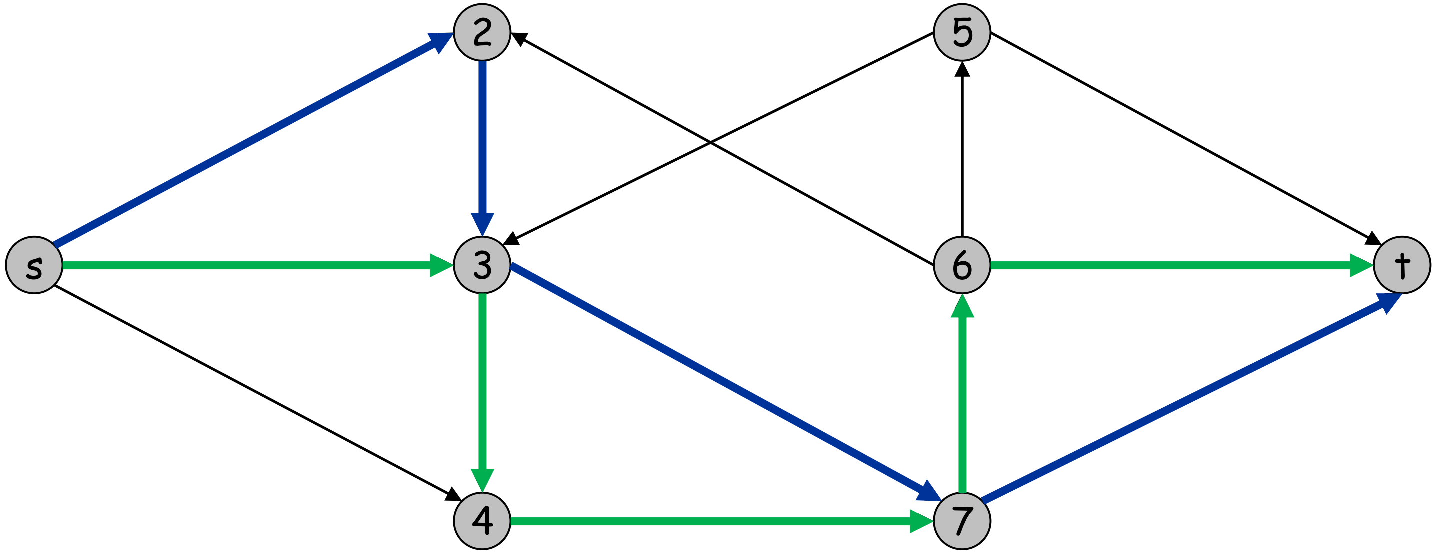
Applications of Max Flow:
Edge Disjoint Paths

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.

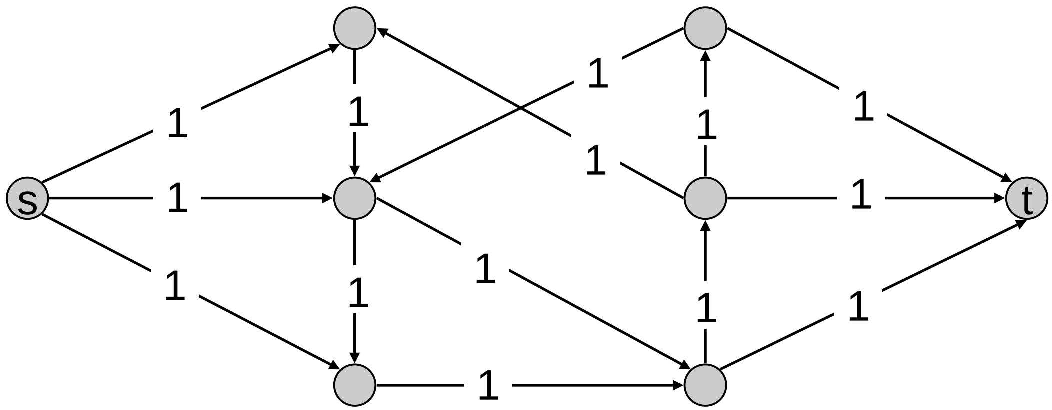
Def. Two paths are edge-disjoint if they have no edge in common.

Ex: communication networks.



Max Flow Formulation

Assign a unit capacitary to every edge. Find Max flow from s to t.



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

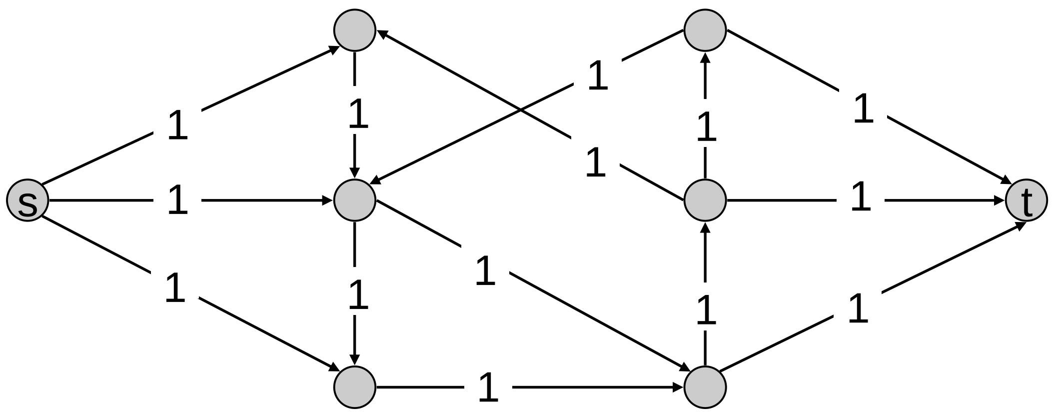
Proof. # of disjoint path ≤ maxflow value

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. ▪

Max Flow Formulation



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

Pf. # of disjoint path ≥ maxflow val 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

This produces k (not necessarily simple) edge-disjoint paths. ▪

We can return to u so we can have cycles. But we can eliminate cycles if desired

Applications of Max Flow:
Project Selection

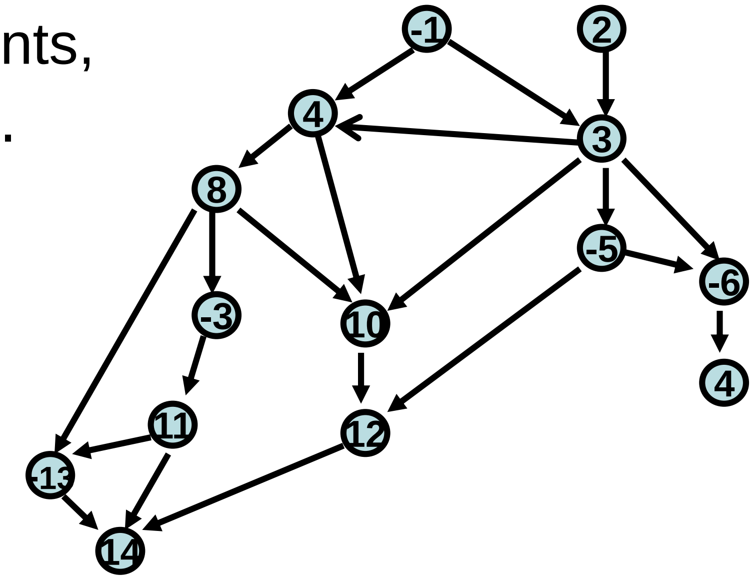
Project Selection

Given a DAG $G = (V, E)$ representing precedence constraints on tasks (a task points to its predecessors).

- Task $v \in V$ has a profit value $p(v)$ (can be positive or negative).

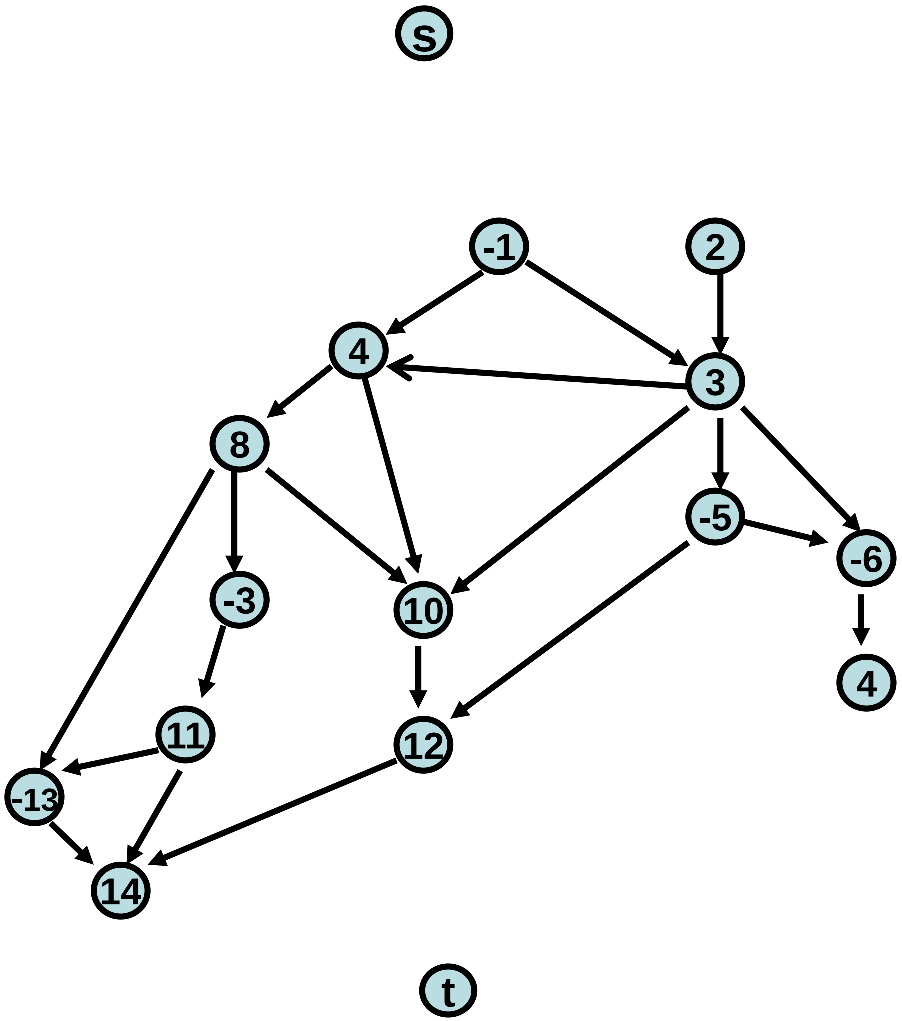
Goal: Find a set $A \subset V$ of tasks that

- satisfies the precedence constraints,
- maximizes $\mathrm{Profit}(A) = \sum_{v \in A} p(v)$.



Each task points to its predecessors

Extended Graph

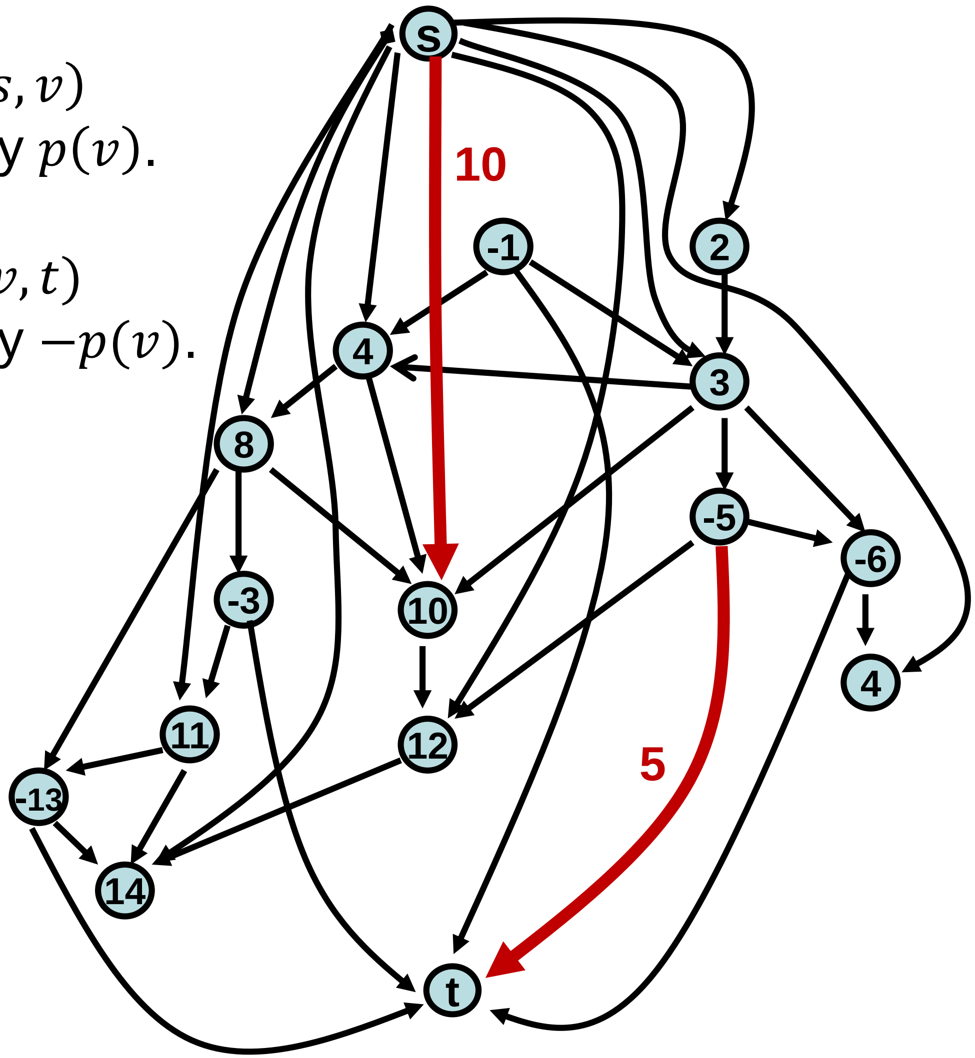


Extended Graph G'

For each v

If $p(v) > 0$, add (s, v) edge with capacity $p(v)$.

If $p(v) < 0$, add (v, t) edge with capacity $-p(v)$.



Extended Graph G'

Goal: Set capacities on edges of G so that
for minimum s-t cut $(S, \bar{S})$ in G', the set $A = S - \{s\}$

- satisfies precedence constraints
- has maximum possible profit in G

To satisfy constraints, don't want any original edges of G cross the minimum cut

- Otherwise, a task in $A = S - \{s\}$ had a predecessor not in A.

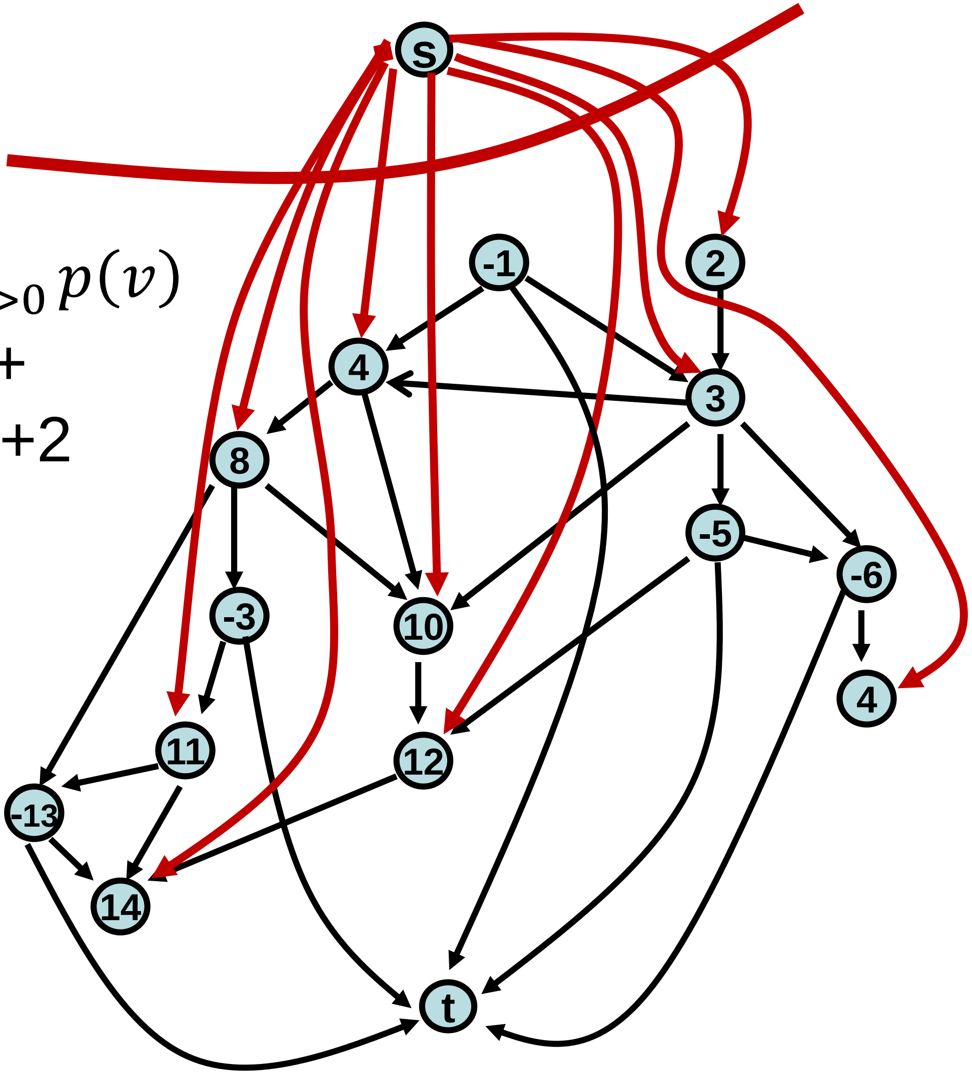
How?

Set capacity of each of the edges of G to $+\infty$.

Extended Graph G'

Cut $C := \sum_{v:p(v)>0} p(v)$

C = 11+8+14+4+
10+12+3+4+2



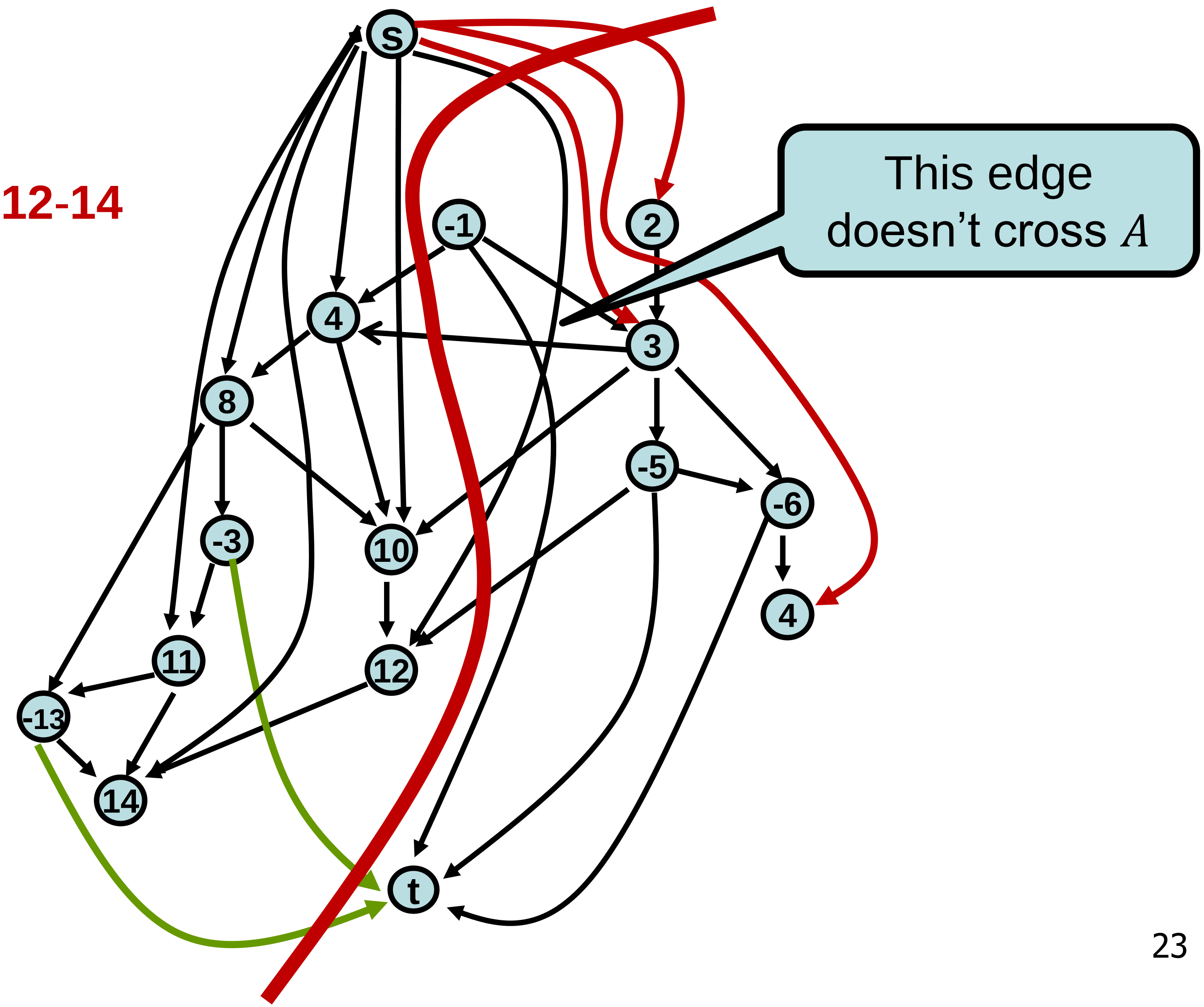
Extended Graph G'

Cut

=13+3+2+3+4

=13+3

+C-4-8-10-11-12-14



Project Selection

Claim: For any s-t cut $(S, \overline{S})$ in G' with finite capacity, the set $A = S - \{s\}$ satisfies

- precedence constraints and
- has capacity $\text{cap}(S, \overline{S}) = C - \sum_{v \in A} p(v) = C - \text{Profit}(A)$

Corollary: A minimum s-t cut $(S, \overline{S})$ in G' yields an optimal solution $A = S - \{s\}$ to the project selection problem

Algorithm:

- Compute maximum flow f in G'
- Find the set S of vertices reachable from s in G'_f
- **Return** $S - \{s\}$

Proof of Claim

- $A = S - \{s\}$ satisfies precedence constraints

No edge of G crosses forward out of A since those edges have capacity $+\infty$

- Capacity = $C - \text{Profit}(A)$

Only forward edges cut are of the form

$$(v, t) \text{ for } v \in A \text{ or } (s, v) \text{ for } v \notin A$$

The (v, t) edges for $v \in A$ contribute

$$\sum_{v \in A : p(v) < 0} -p(v) = -\sum_{v \in A : p(v) < 0} p(v)$$

The (s, v) edges for $v \notin A$ contribute

$$\sum_{v \notin A : p(v) > 0} p(v) = C - \sum_{v \in A : p(v) > 0} p(v)$$

Therefore, the total capacity is

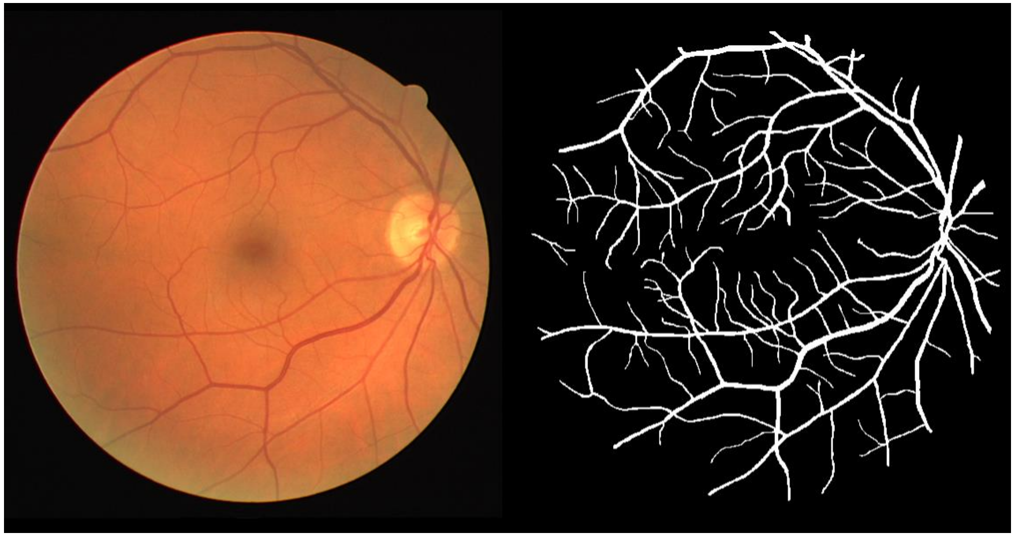
$$C - \sum_{v : p(v) > 0} p(v) = C - \text{Profit}(A)$$

Applications of Max Flow: Image Segmentation

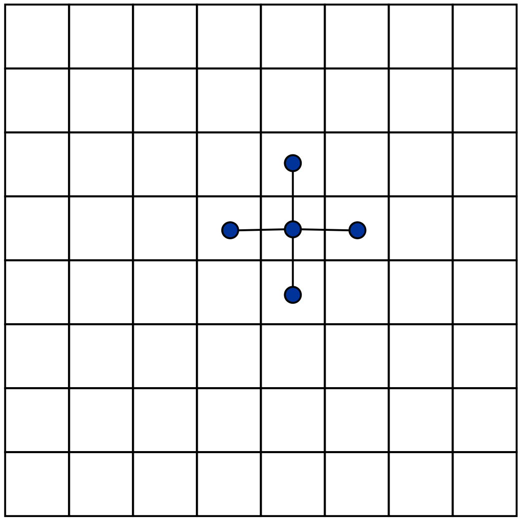
Image Segmentation

Given an image we want to separate foreground from background

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



Foreground / background segmentation



Label each pixel as foreground/background.

- V = set of pixels, E = pairs of neighboring pixels.
- a_i is the original image.
- $a_i \gg 0$ means we prefer to label i in foreground.
- $p_{i,j} \geq 0$ is separation penalty for labeling one of i and j as foreground, and the other as background.

Goals:

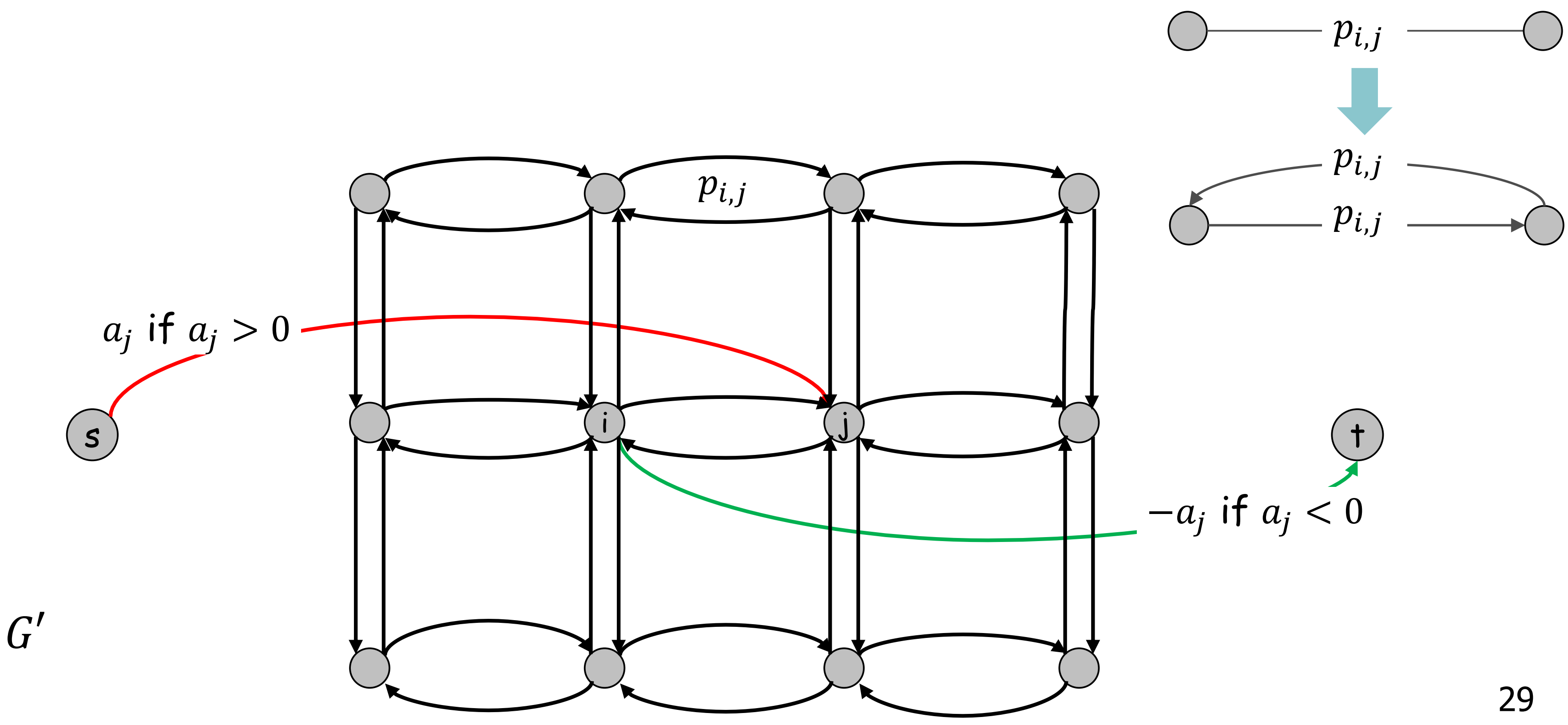
Find partition $(S, \overline{S})$ that minimizes:

$$-\sum_{i \in S} a_i + \sum_{\substack{(i,j) \in E \\ i \in S, j \in \overline{S}}} p_{i,j}$$

where S is the foreground.

Min cut Formulation

$G' = (V', E')$.
Add s to correspond to foreground;
Add t to correspond to background;
Use two anti-parallel edges instead of undirected edge.



Min cut Formulation (cont'd)

- Consider min cut $(S, \overline{S})$ in G'. (S = foreground.)

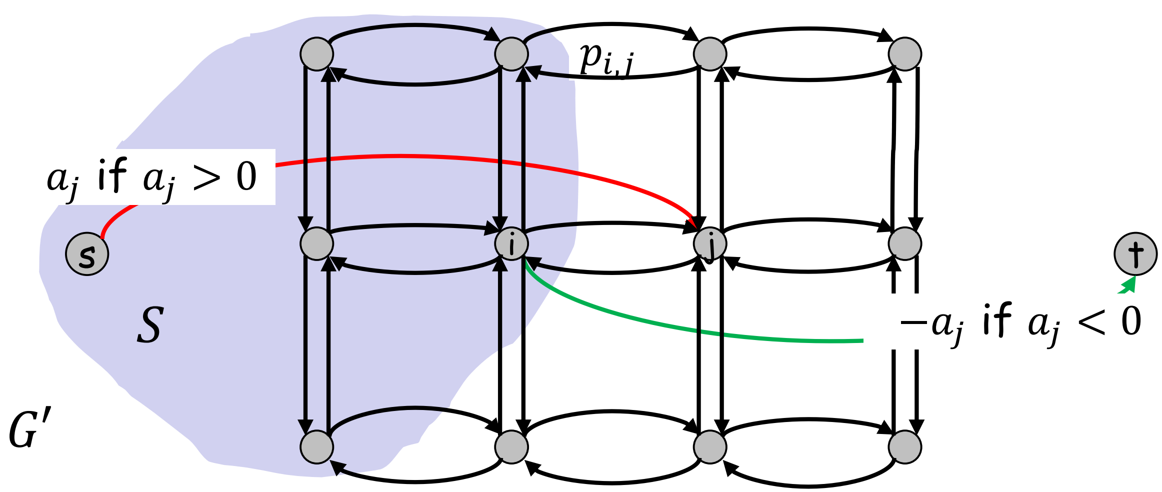
$$cap(S, \overline{S}) = \sum_{i \in S} -a_i \, 1_{a_i < 0} + \sum_{i \in \overline{S}} a_i \, 1_{a_i > 0} + \sum_{\substack{(i,j) \in E \\ i \in S, j \in \overline{S}}} p_{i,j}$$

$$= -\sum_{i \in S} a_i + \sum_{i \in S} a_i \, 1_{a_i > 0} + \sum_{i \in \overline{S}} a_i \, 1_{a_i > 0} + \sum_{\dots} p_{i,j}$$

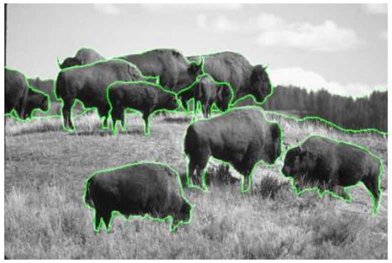
$$= -\sum_{i \in S} a_i + \sum_i a_i + \sum_{\dots} p_{i,j}$$

$$= -\sum_{i \in S} a_i + \sum_{\dots} p_{i,j} + \text{constant}$$

Precisely, what we want to minimize.

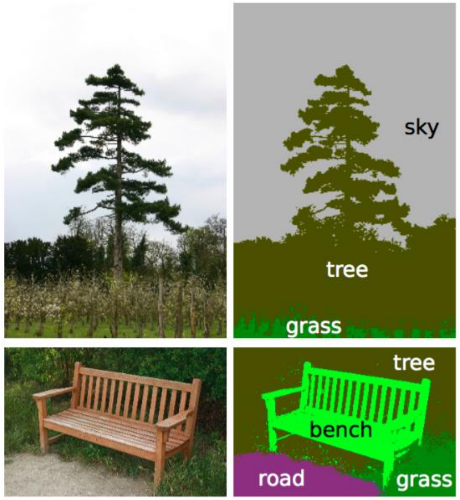


Remark



- The main difficulty is to come up with a good model.
- May want to have human interaction.

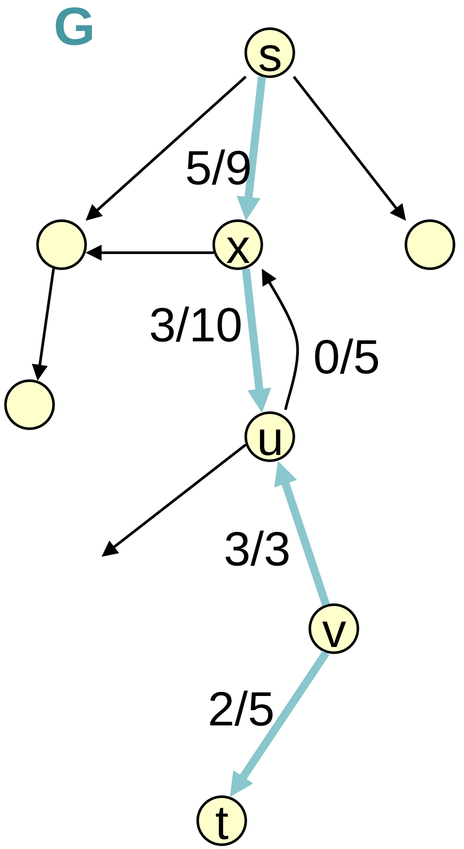
- Segmentation may be real-valued instead of {0,1}.
- There are many more than 1 objects.
- May need labeling.
- Augmenting path is not great for GPU.

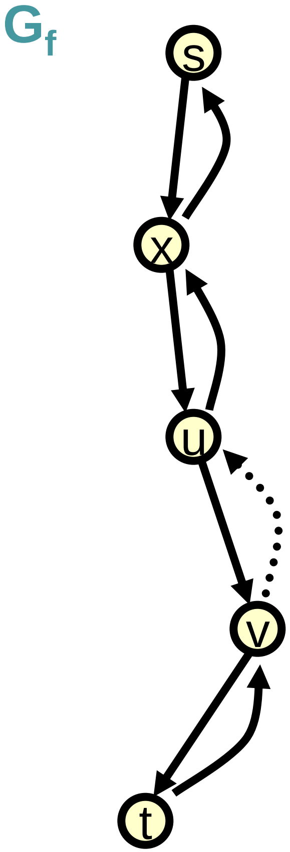


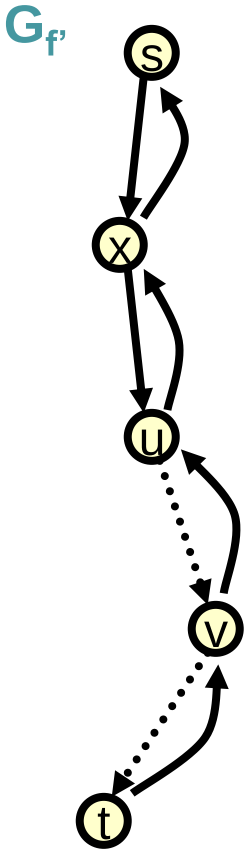
Edmonds-Karp Algorithm

Edmonds-Karp Algorithm

- Use a shortest augmenting path
  (via Breadth First Search in residual graph)
- Time: $O(m^2 n)$.







Distance to s is non-decreasing.

Let f be a flow, G_f the residual graph, and P a shortest augmenting path. Then no vertex is closer to s after augmentation along P.

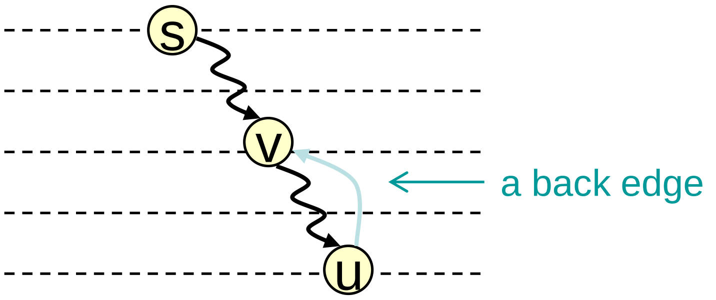
Proof: Augmentation along P only

- deletes forward edges

  no new (hence no shorter) path created

- adds back edges that go to previous vertices along P

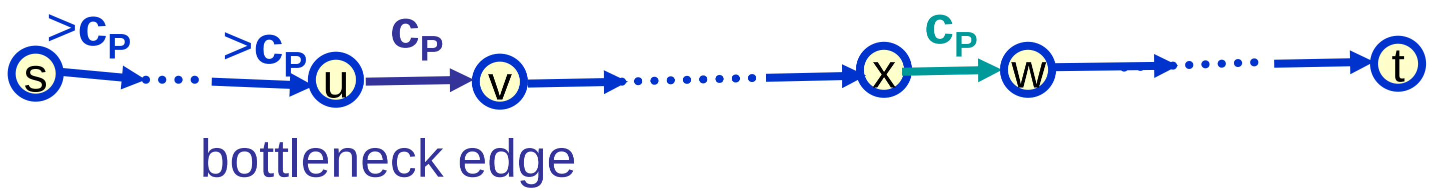
  BFS is unchanged, since v visited before (u, v) examined



Distance for bottleneck edges

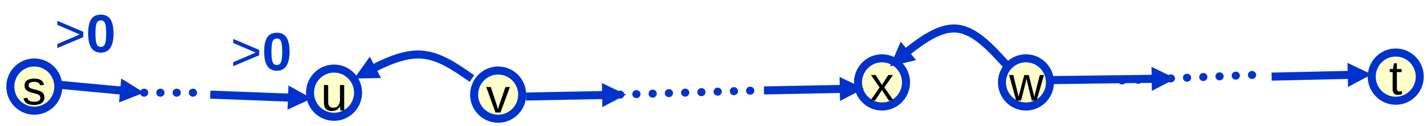
Let $d_f(s,v)$ be the distance from s to v on G_f.

Shortest s-t path **P** in **G$_f$**

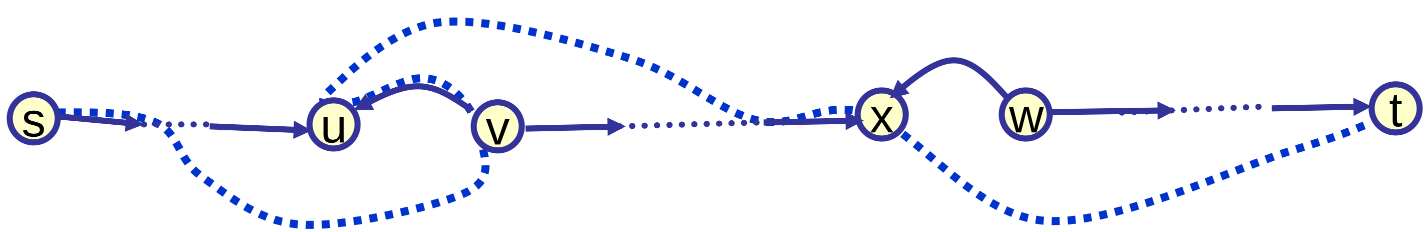


After augmenting along P

$d_f(s,v) = d_f(s,u) + 1$ since this is a shortest path



For (u,v) to be bottleneck again for some flow f'



$$d_{f'}(s,u) = d_{f'}(s,v) + 1 \geq d_f(s,v) + 1 = d_f(s,u) + 2$$

Theorem

Edmonds-Karp performs $O(mn)$ flow augmentations

Proof:

- Each step, some edge disappear from G_f.
  (Note however that some edge may reappear.)
- Any edge (u, v) disappears from G_f at most $n/2$ times.
  (because the distance increased by 2 every disappearance.)
- There are at most $mn/2$ disappearances.

Total time is $O(m^2n)$.