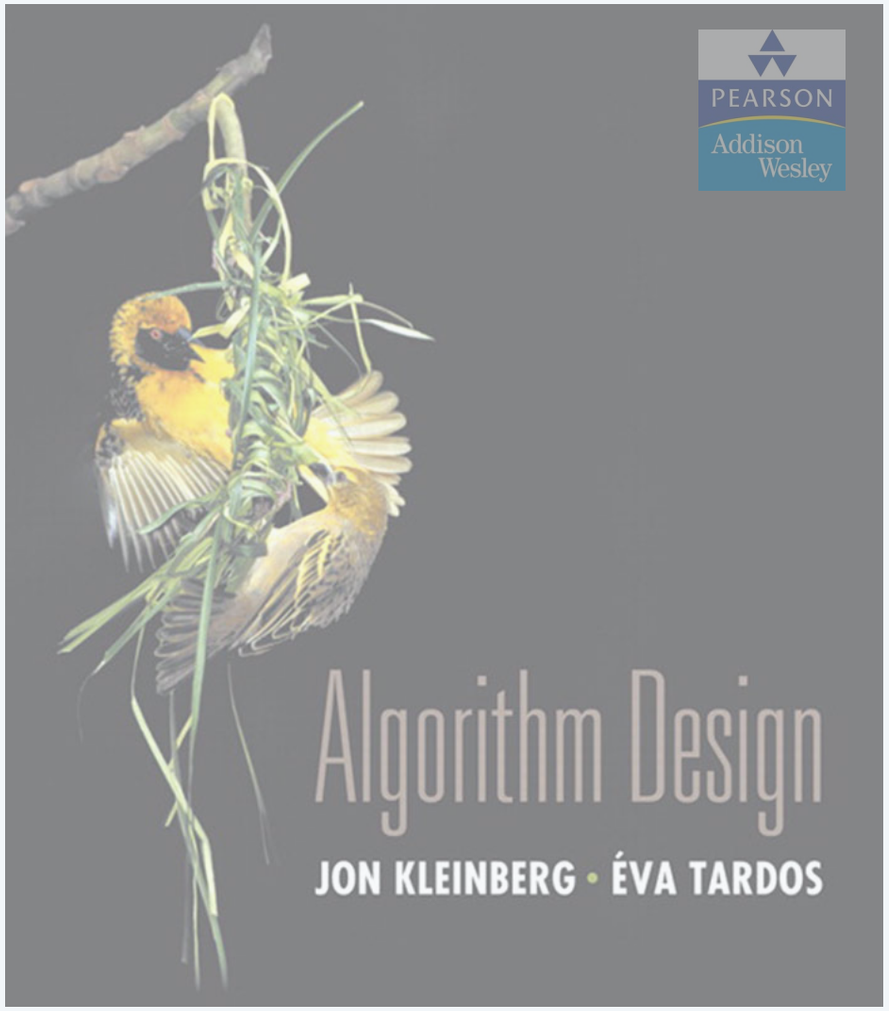
23/10/21



SECTION 7.7

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Network flow II: quiz 4

Which extensions to max flow can be easily modeled?

A. Multiple sources and multiple sinks.

B. Undirected graphs.

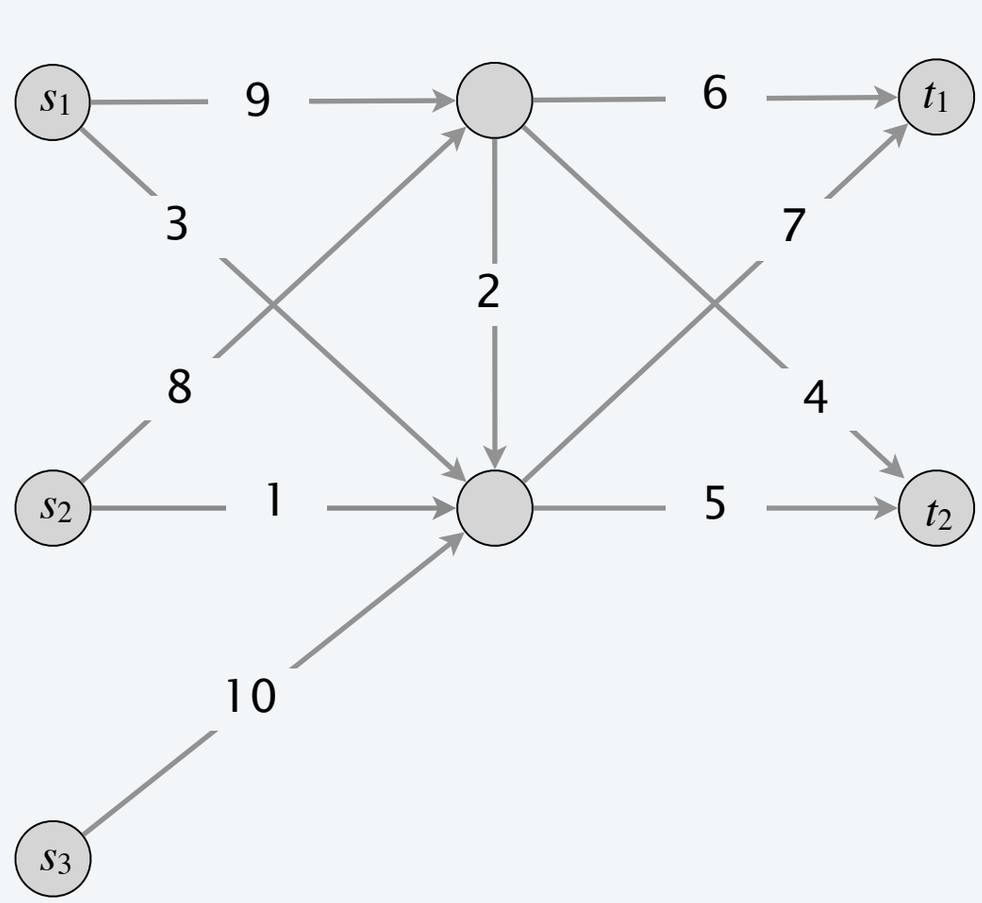
C. Lower bounds on edge flows.

D. All of the above.

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: max-flow formulation

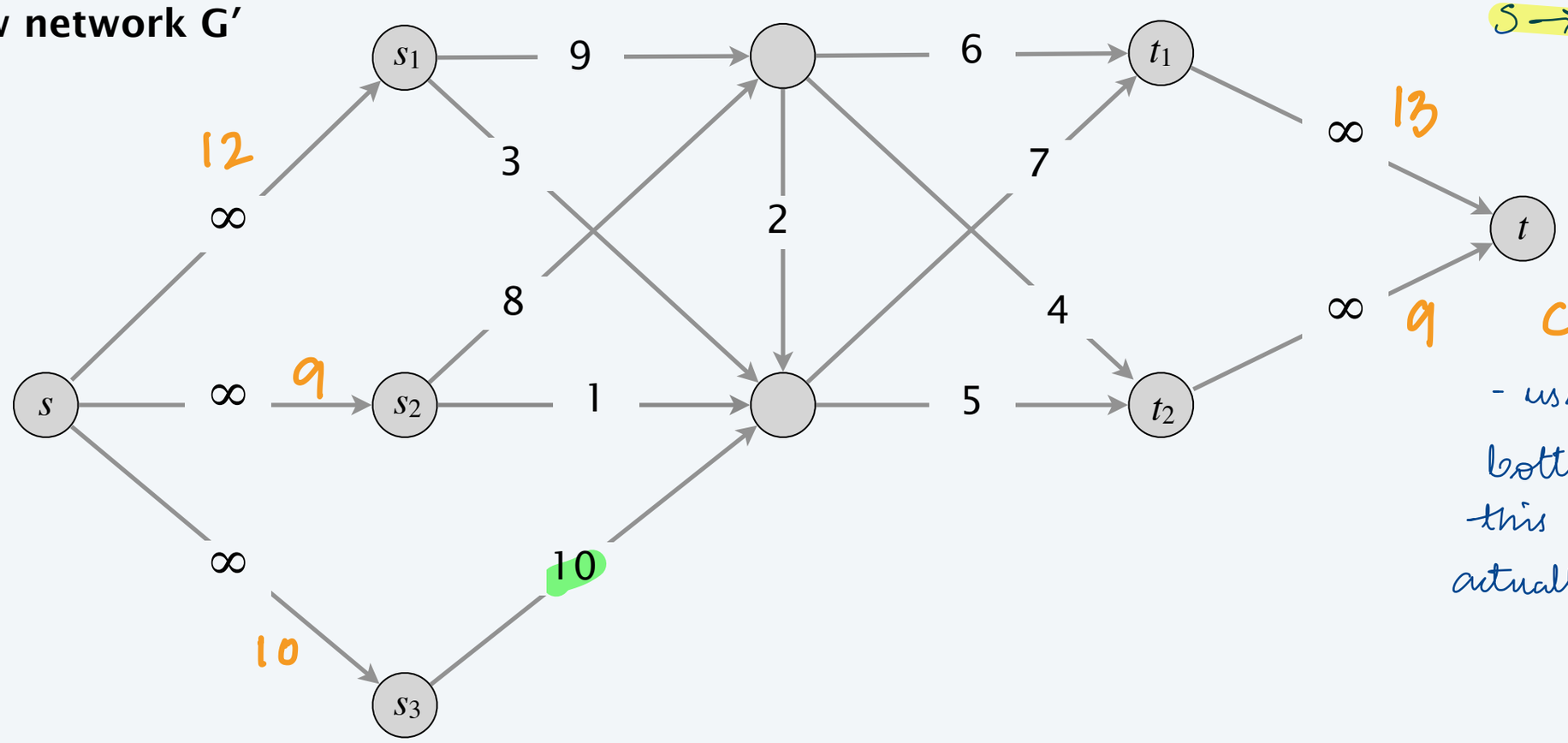
- Add a new source node s and sink node t.
- For each original source node s_i add edge (s, s_i) with capacity ∞.
- For each original sink node t_j, add edge (t_j, t) with capacity ∞.

Claim. 1–1 correspondence between flows in G and G'.

App 1:
In practical approach adding '∞' is not possible, thus we add $S \to S_i$, sum of outgoing edges from S_i

C Mistake:
- using some imaginary bottleneck say in this case 10, which actually won't work!

flow network G'

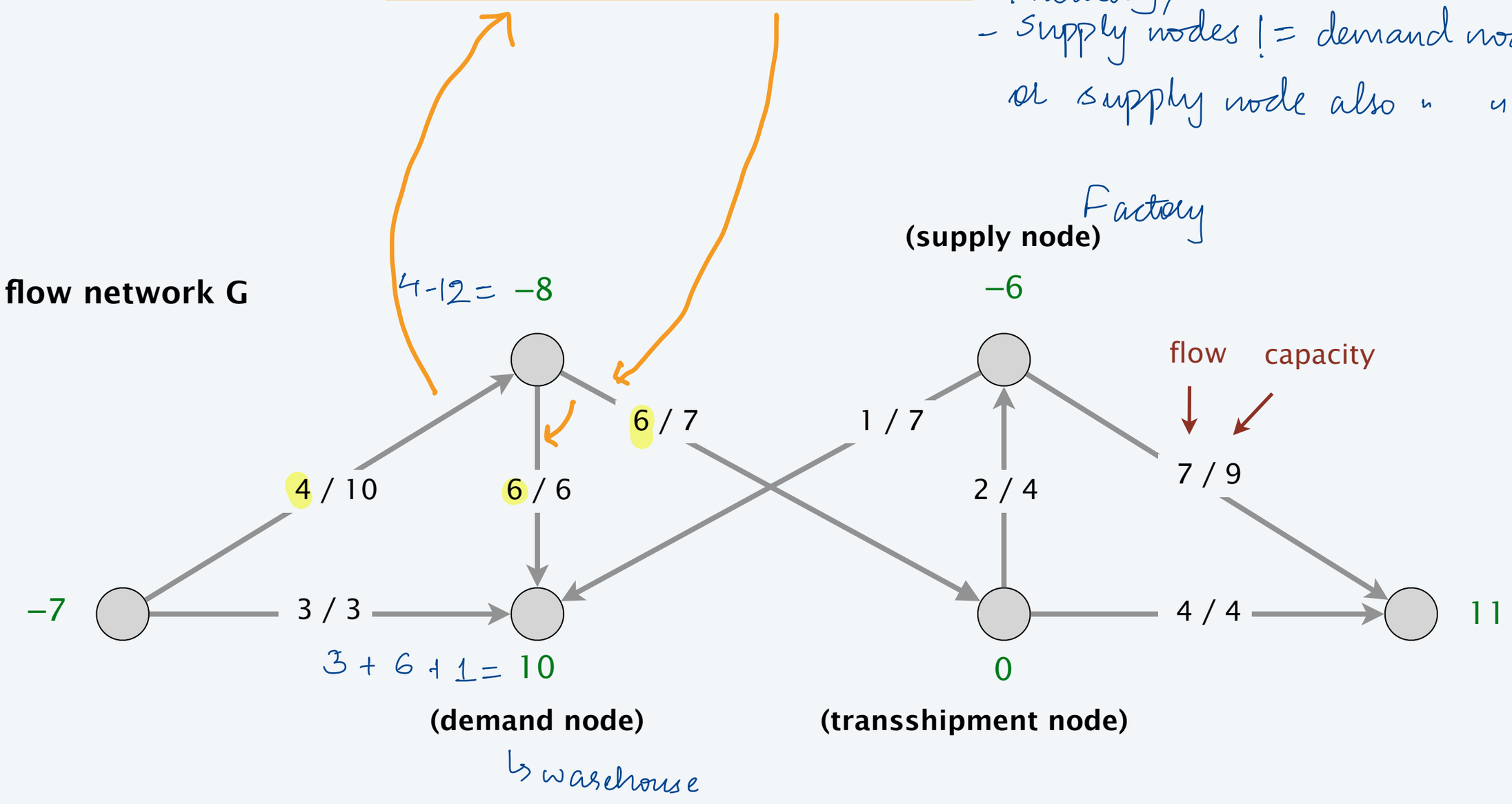


Circulation with supplies and demands

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 $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (flow conservation)

Generally,
- supply nodes != demand nodes
or supply node also " "



Circulation with supplies and demands: max-flow formulation

↳ flow shows that we can meet all the demands!

- 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 $d(v)$.

– This creates a complete Production cycle

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

saturates all edges leaving s and entering t

P5 : S → A → E → B → F → t (1) } Paths not in Graph 3
P6 : S → A → E → 3 → D → t (1)

Max-Flow = '21'

flow network G'

supply

we pretend this comes from 's'

P1 → S → A → D → t, bottle-6
P2 → S → B → E → t, bott = 6

P3 : S → C → D → t
b = 3

P4 : S → C → A
11 ↓
t ← F ← E
b = 4

demand

Value of the flow!

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 $\Sigma_{v \in B} d(v) > cap(A, B)$.

Pf sketch. 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 supplies, demands, and lower bounds

↗ Threshold below which edge will not be used!
eg. Sending UPS truck with single item!

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 $f(e)$ is a function that satisfies:

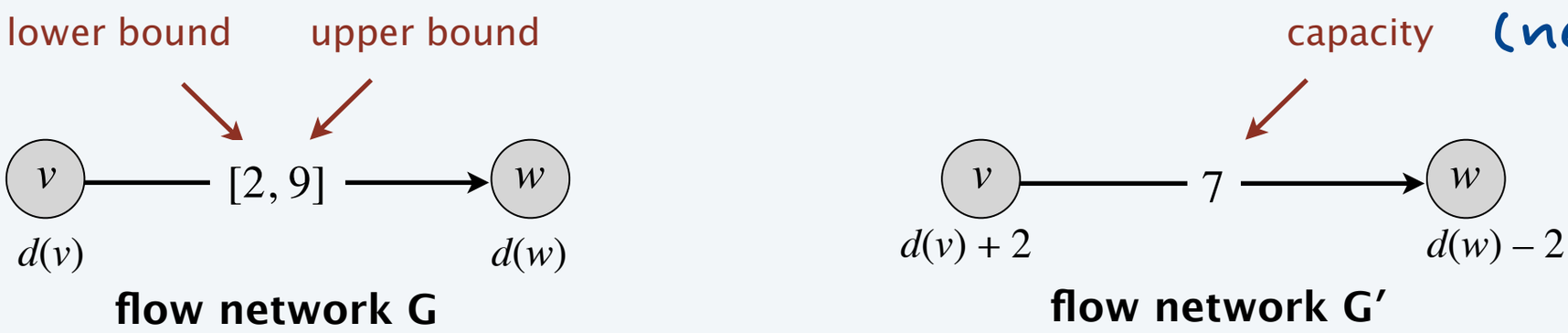
- For each $e \in E$: $\ell(e) \leq f(e) \leq c(e)$ (capacity)
- For each $v \in V$: $\sum_{e \text{ in to } v} f(e) - \sum_{e \text{ out of } v} f(e) = d(v)$ (flow conservation)

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

Circulation with supplies, demands, and lower bounds

Max-flow formulation. Model lower bounds as circulation with demands.

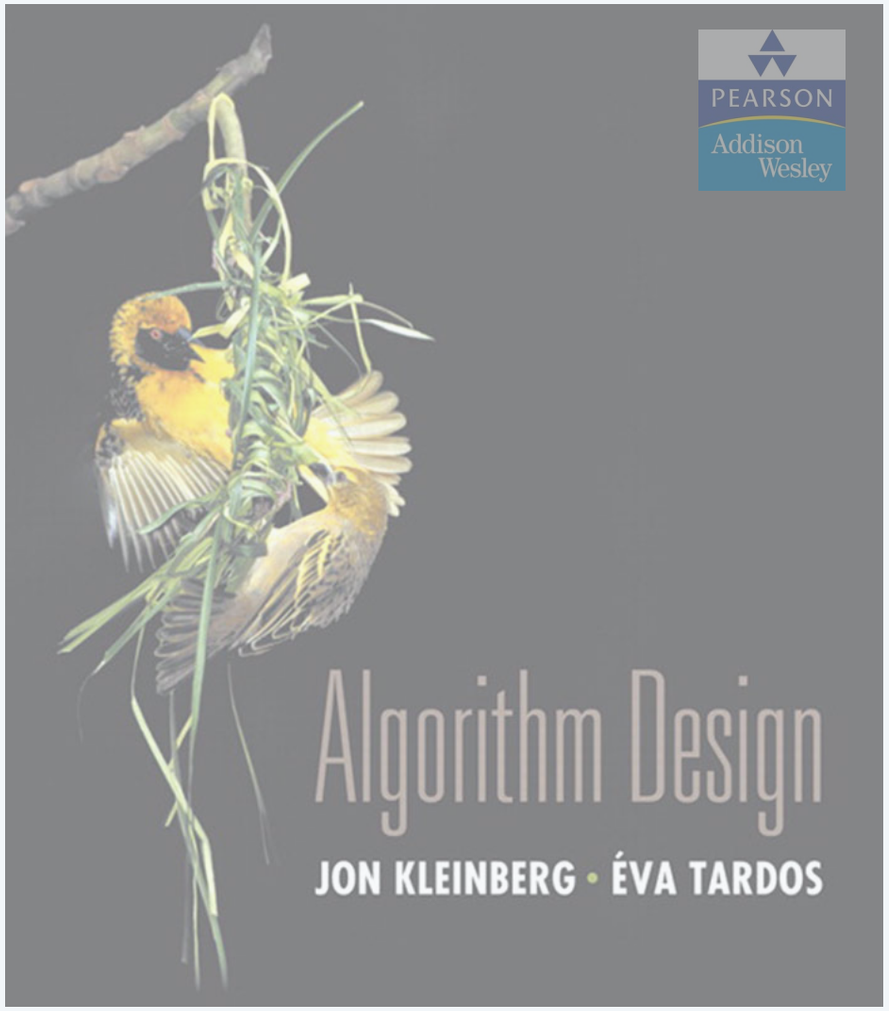
- Send $\ell(e)$ units of flow along edge e.
- Update demands of both endpoints.



flow network G **flow network 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'.



SECTION 7.8

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Survey design

- Design survey asking n_1 consumers about n_2 products. ← one survey question per product
- Can survey consumer i about product j only if they own it.
- Ask consumer i between c_i and c_i' questions. → to make sure single customer is not asked to many Qs!
- Ask between p_j and p_j' consumers about product j. → to ensure every pdt has a min. surveys.

lower Bound upper Bound

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

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

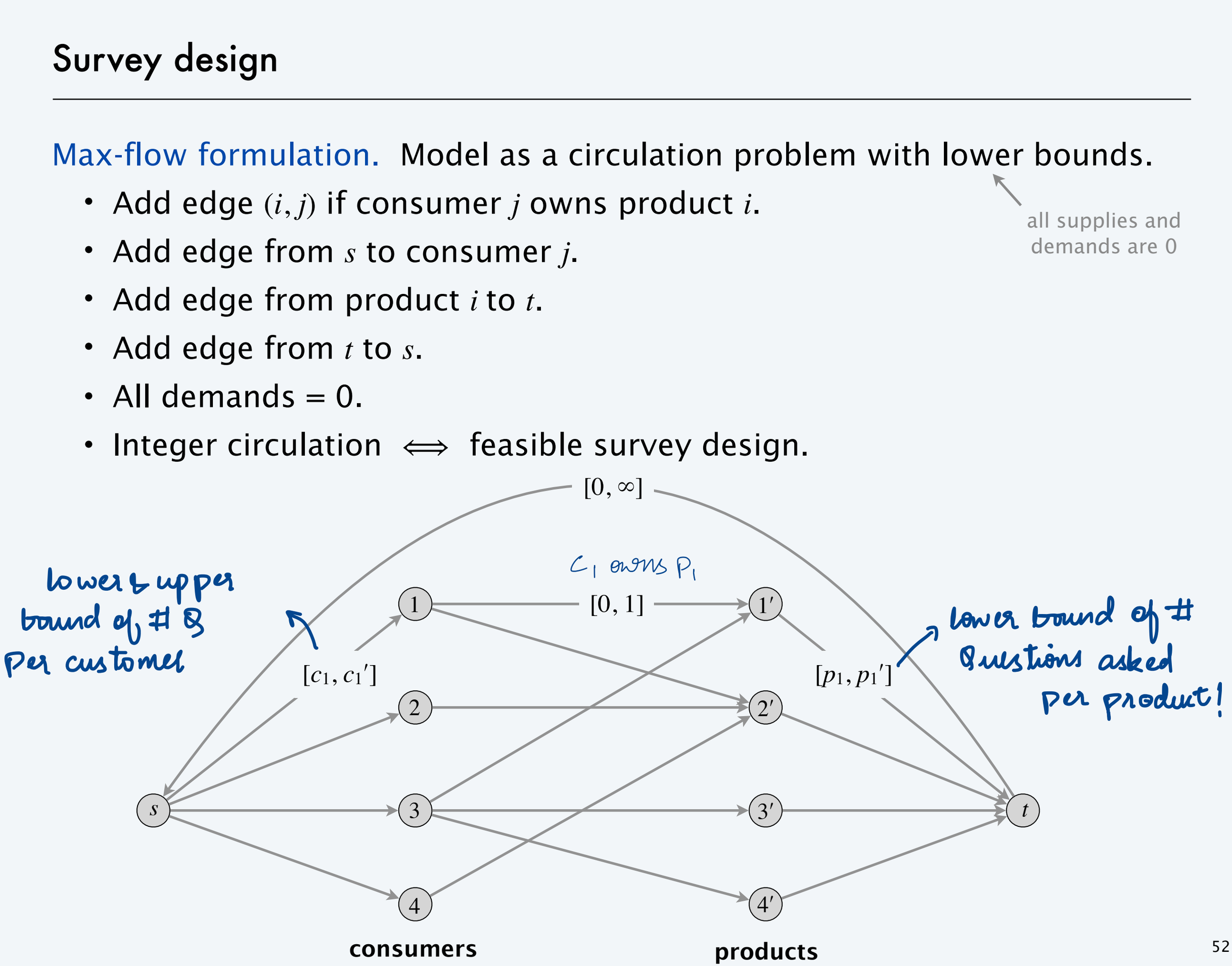
We won't ask all the quest^n to C3 ... Cn, too many Questions

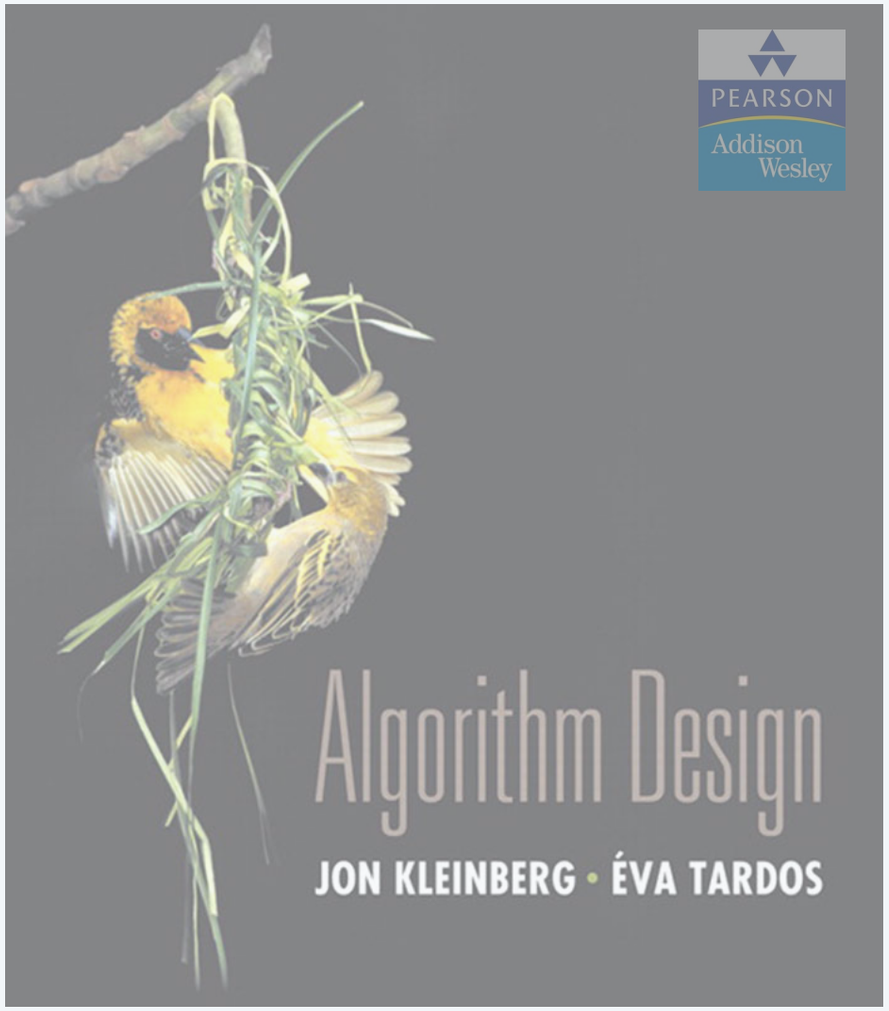
Survey design

Max-flow formulation. Model as a circulation problem with lower bounds.

- Add edge (i, j) if consumer j owns product i.
- Add edge from s to consumer j.
- Add edge from product i to t.
- Add edge from t to s.
- All demands = 0.
- Integer circulation $\iff$ feasible survey design.

all supplies and demands are 0





SECTION 7.9

7. NETWORK FLOW II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- ✓ *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

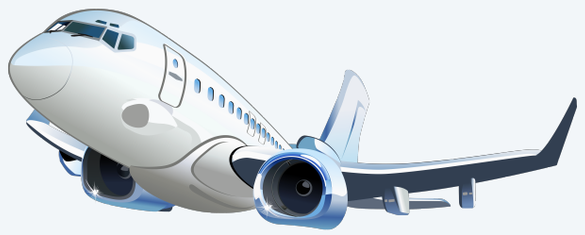
Airline scheduling

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. ← even in presence of unpredictable events, such as weather and breakdowns
- One of largest consumers of high-powered algorithmic techniques.

"Toy problem."

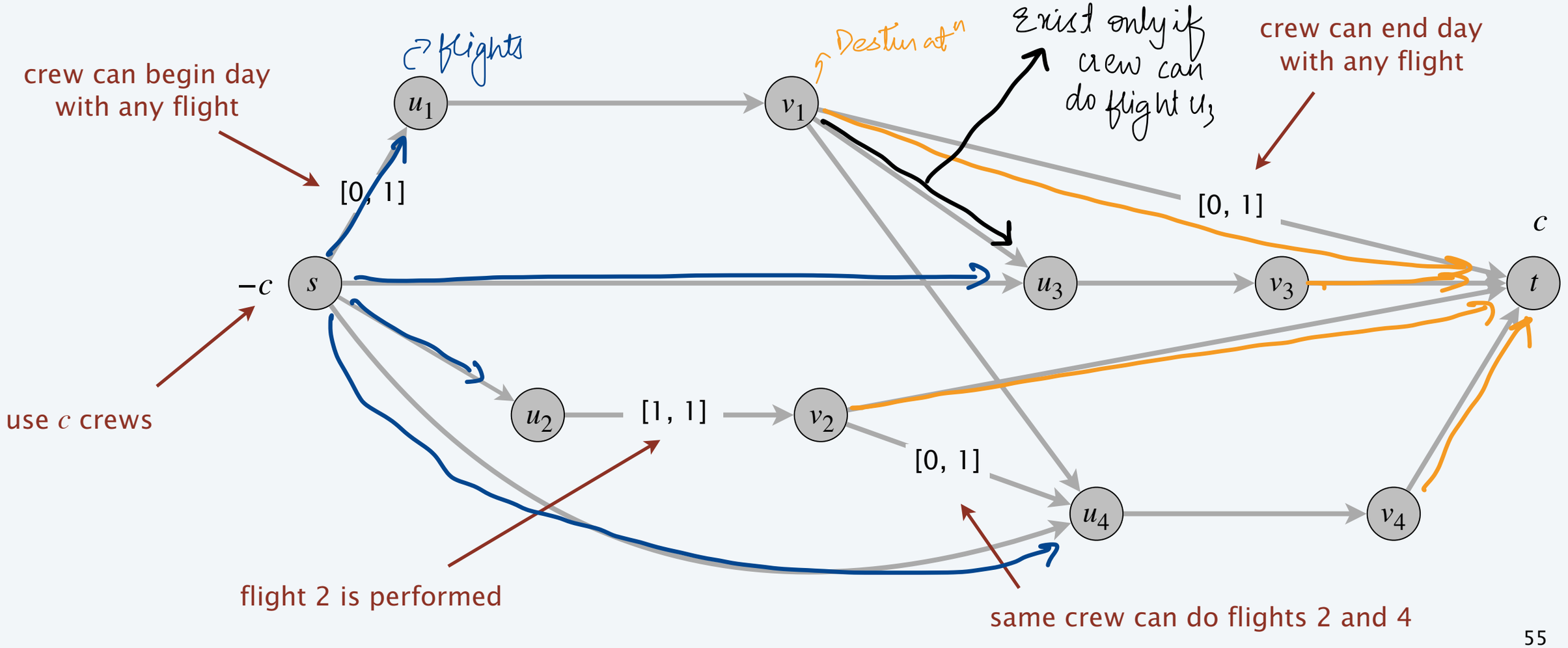
- Manage flight crews by reusing them over multiple flights.
- Input: set of k flights for a given day.
- Flight i leaves origin o_i at time s_i and arrives at destination d_i at time f_i.
- Minimize number of flight crews.



Airline scheduling

Circulation formulation. [to see if c crews suffice]

- For each flight i, include two nodes u_i and v_i. ← u_i = start of flight i, v_i = end of flight i
- 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.
- For each i, add edge (u_i, v_i) with lower bound and capacity 1.
- if flight j reachable from i, add edge (v_i, u_j) with capacity 1.



Airline scheduling: running time

Theorem. The airline scheduling problem can be solved in $O(k^3 \log k)$ time.

Pf.

- k = number of flights.
- c = number of crews (unknown).
- $O(k)$ nodes, $O(k^2)$ edges. — 2K + 2 | 2 (ui, vi) ∀ 's' & 't'
- At most k crews needed. — every crew on edge!

  $\Rightarrow$ solve $\log_2 k$ circulation problems. ← binary search for min value c^*

- Value of any flow is between 0 and k.

  $\Rightarrow$ at most k augmentations per circulation problem.

- Overall time = $O(k^3 \log k)$. Ford-Fulkerson!

  ⟶ $O(mC) \times O(\log_2 K)$

  K^2 ↳ K

Remark. Can solve in $O(k^3)$ time by formulating as minimum-flow problem.

Airline scheduling: postmortem

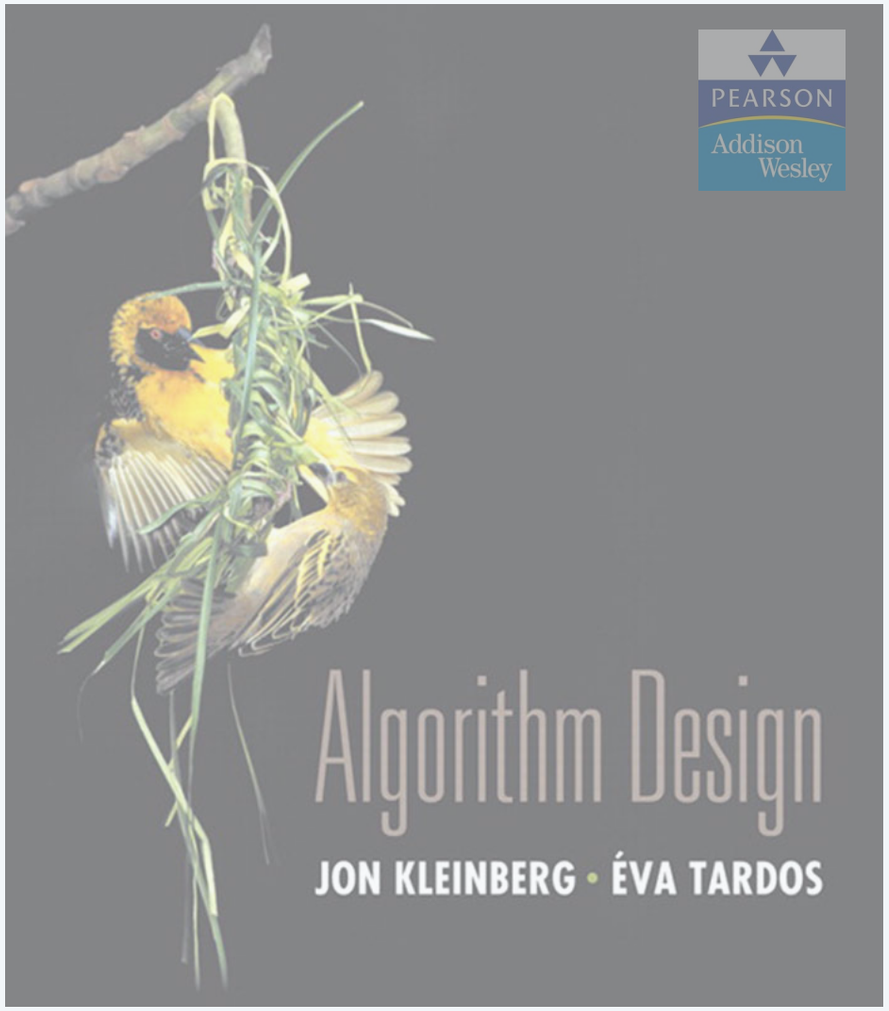
Remark. We solved a toy version of a real problem.

Real-world problem models countless other factors:

- Union regulations: e.g., 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.
- Deadheading has a cost.
- 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.



SECTION 7.10

7. NETWORK FLOW II

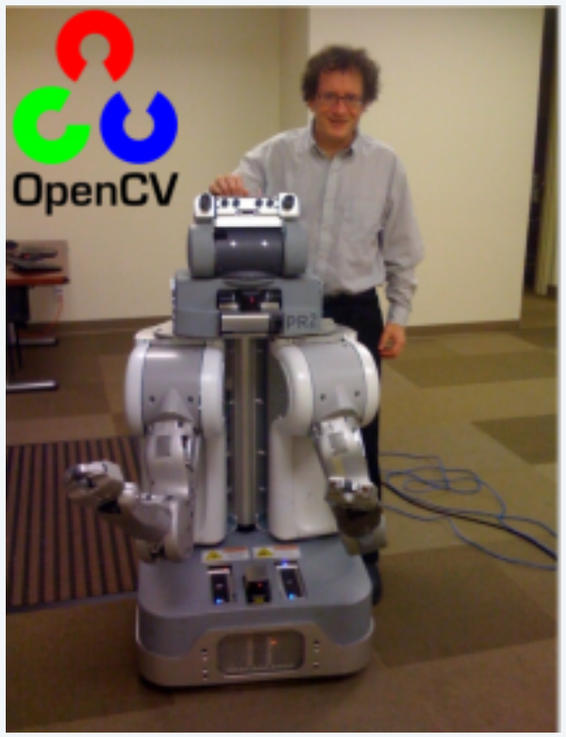
- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

Image segmentation

Image segmentation.

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

Ex. Separate human and robot from background scene.



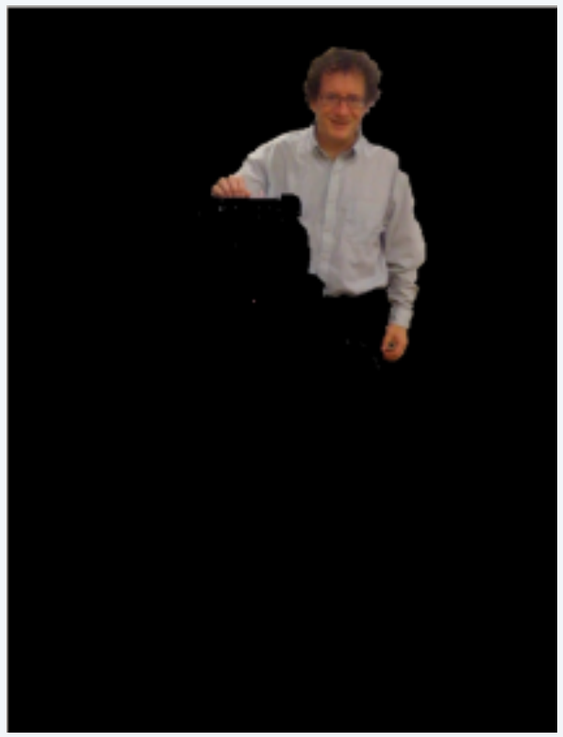
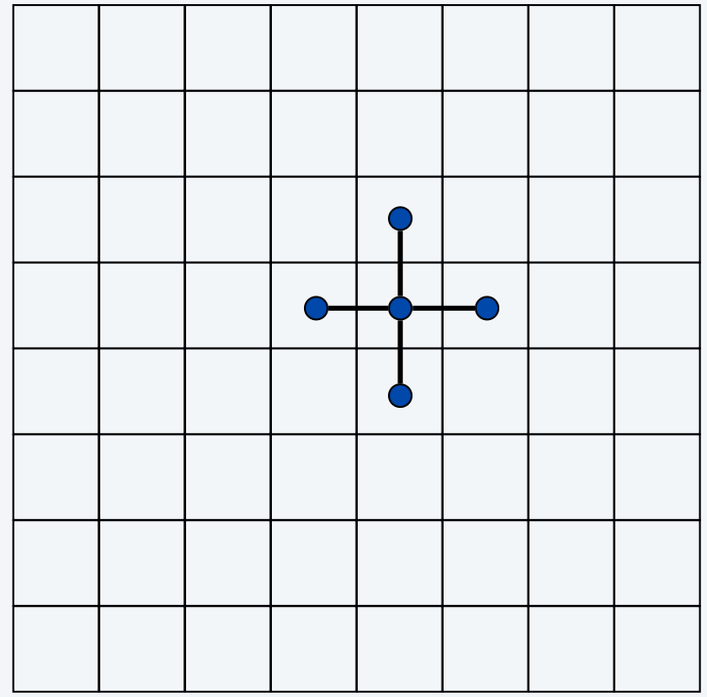


Image segmentation

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.
- $p_{ij} \geq 0$ is separation penalty for labeling one of i and j as foreground, and the other as background.

$(i,j) \in E$ only for neighbours.

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

foreground (A) background (B)

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

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

a constant

- or alternatively

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

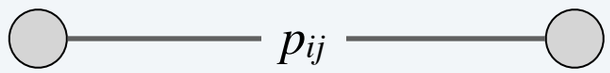
means only 1 element in $A \cap \{i,j\}$ ⇔ either i/j is in 'A' not both/ not 0.

Image segmentation e.g. in Notes!

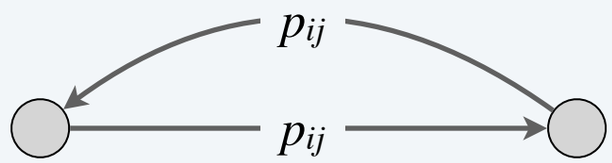
Formulate as min-cut problem $G' = (V', E')$.

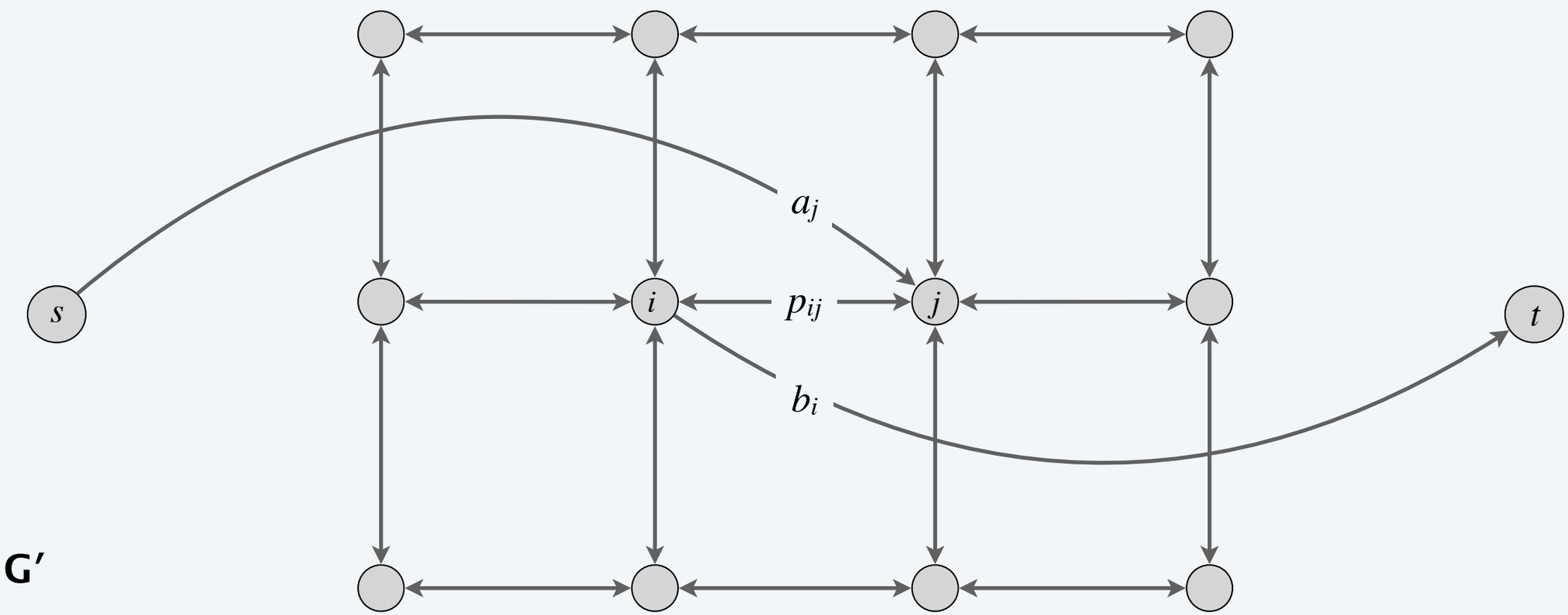
- Include node for each pixel. $s \xrightarrow{a_i} i \xrightarrow{b_i} t$
- Use two antiparallel edges instead of undirected edge.
- Add source s to correspond to foreground.
- Add sink t to correspond to background.

edge in G



two antiparallel edges in G′





G′

Image segmentation

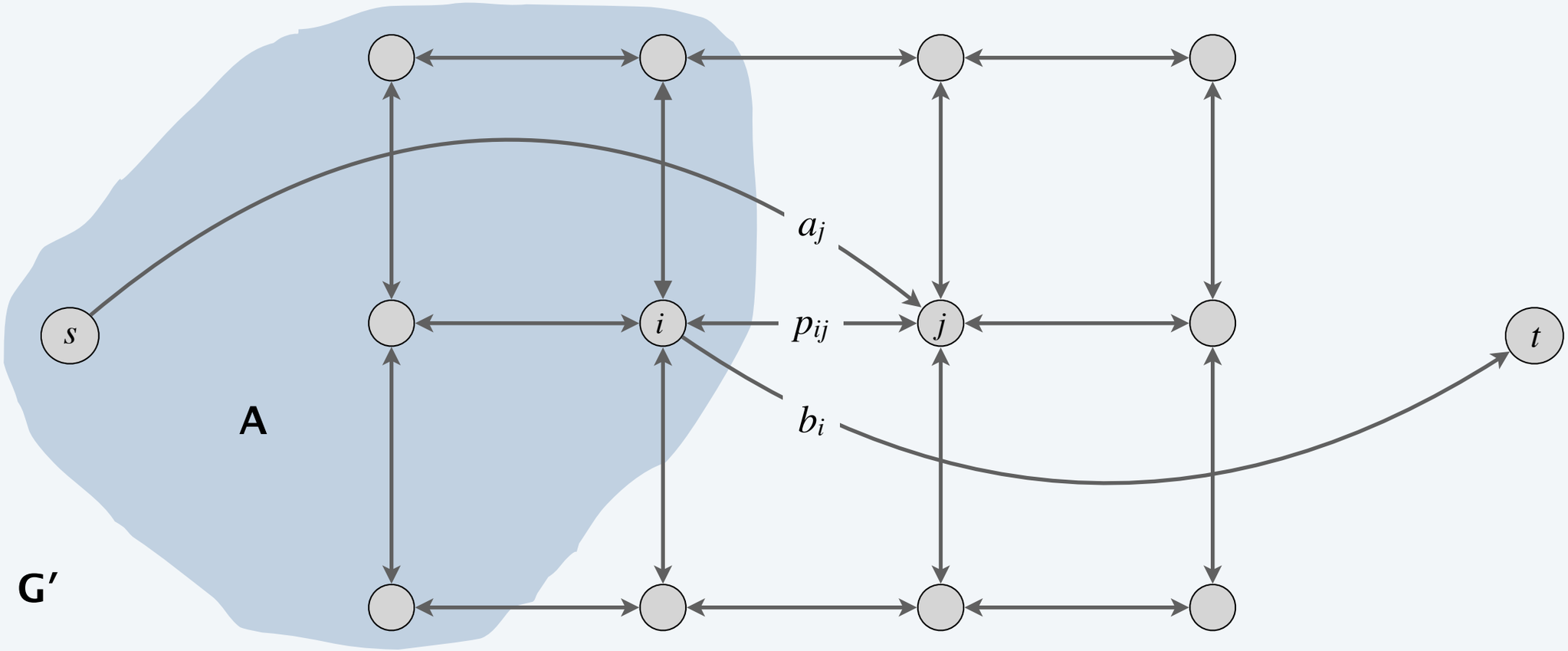
Consider min cut (A, B) in G'. ∫– tells us what goes in forward!

- A = foreground.

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

← if i and j on different sides, p_{ij} counted exactly once

- Precisely the quantity we want to minimize.



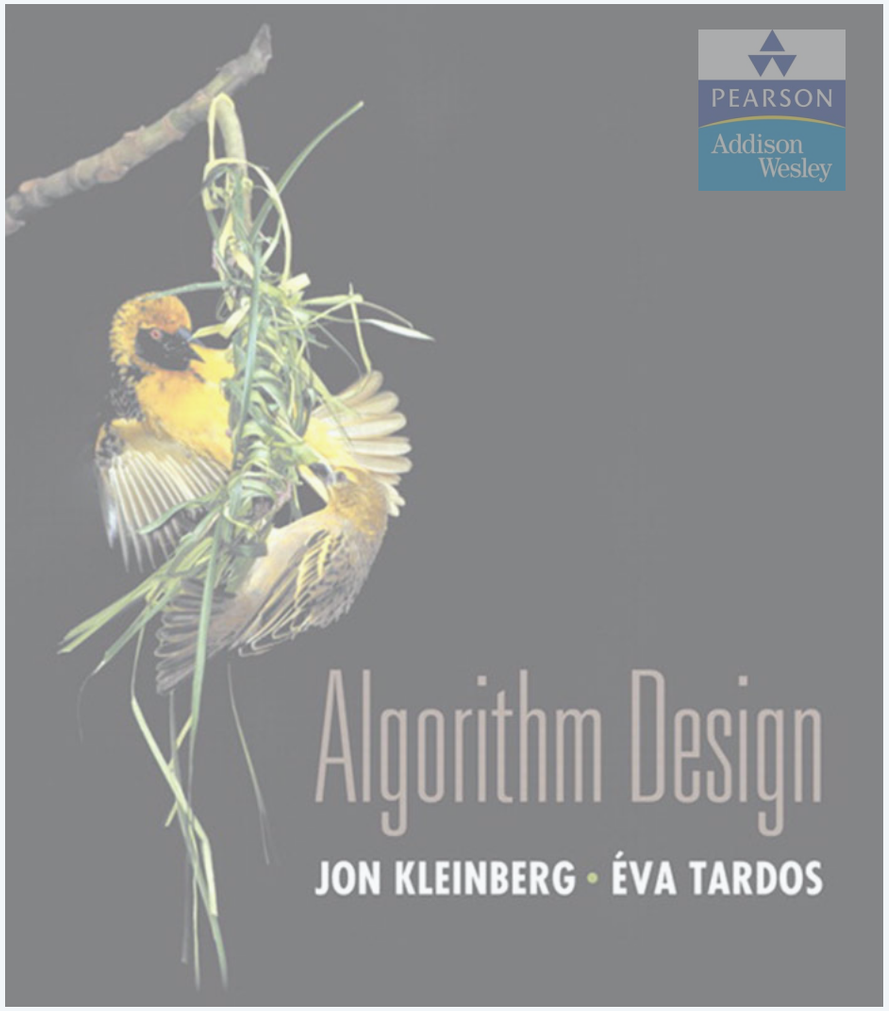
Grabcut image segmentation

Grabcut. [Rother–Kolmogorov–Blake 2004]

"GrabCut" — Interactive Foreground Extraction using Iterated Graph Cuts

Carsten Rother* Vladimir Kolmogorov† Andrew Blake‡
Microsoft Research Cambridge, UK

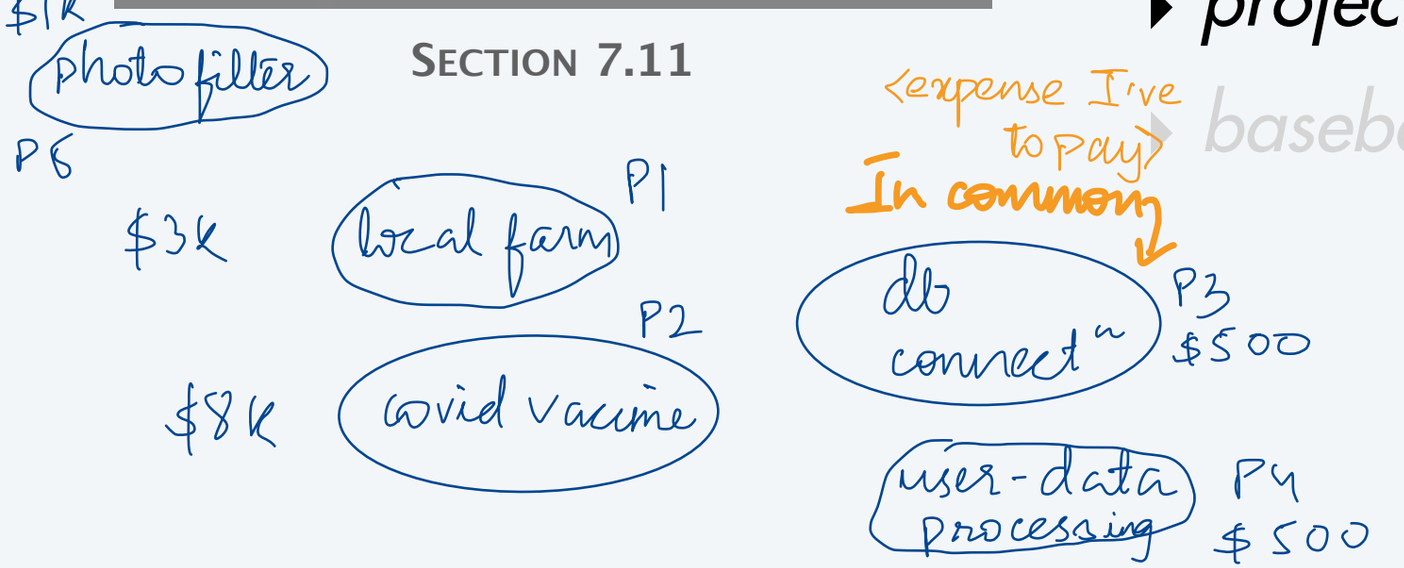
Figure 1: **Three examples of GrabCut .** The user drags a rectangle loosely around an object. The object is then extracted automatically.



7. Network Flow II

- *bipartite matching*
- *disjoint paths*
- *extensions to max flow*
- *survey design*
- *airline scheduling*
- *image segmentation*
- *project selection*
- *baseball elimination*

SECTION 7.11



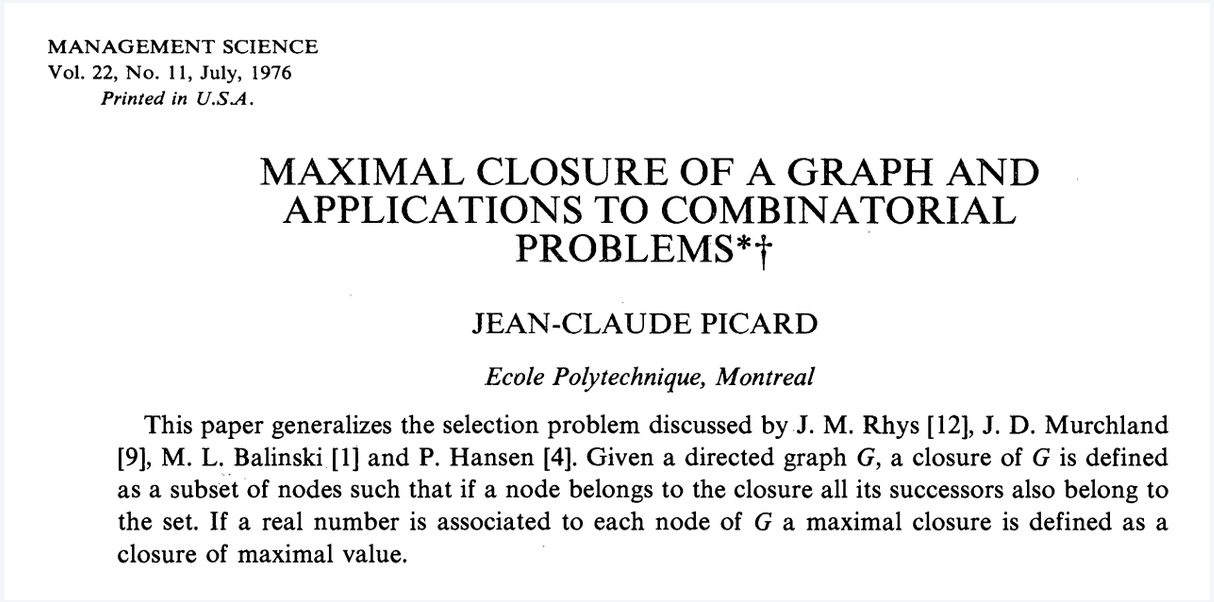
Project selection (maximum weight closure problem)

Projects with prerequisites.

- Set of possible projects P: project v has associated revenue p_v. (can be positive or negative)
- Set of prerequisites E: $(v, w) \in E$ means w is a prerequisite for v.
- A subset of projects $A \subseteq P$ is feasible if the prerequisite of every project in A also belongs to A.

↳ for P1, P3 & P4 needs to be selected!

Project selection problem. Given a set of projects P and prerequisites E, choose a feasible subset of projects to maximize revenue.



MANAGEMENT SCIENCE
Vol. 22, No. 11, July, 1976
Printed in U.S.A.

MAXIMAL CLOSURE OF A GRAPH AND APPLICATIONS TO COMBINATORIAL PROBLEMS*†

JEAN-CLAUDE PICARD

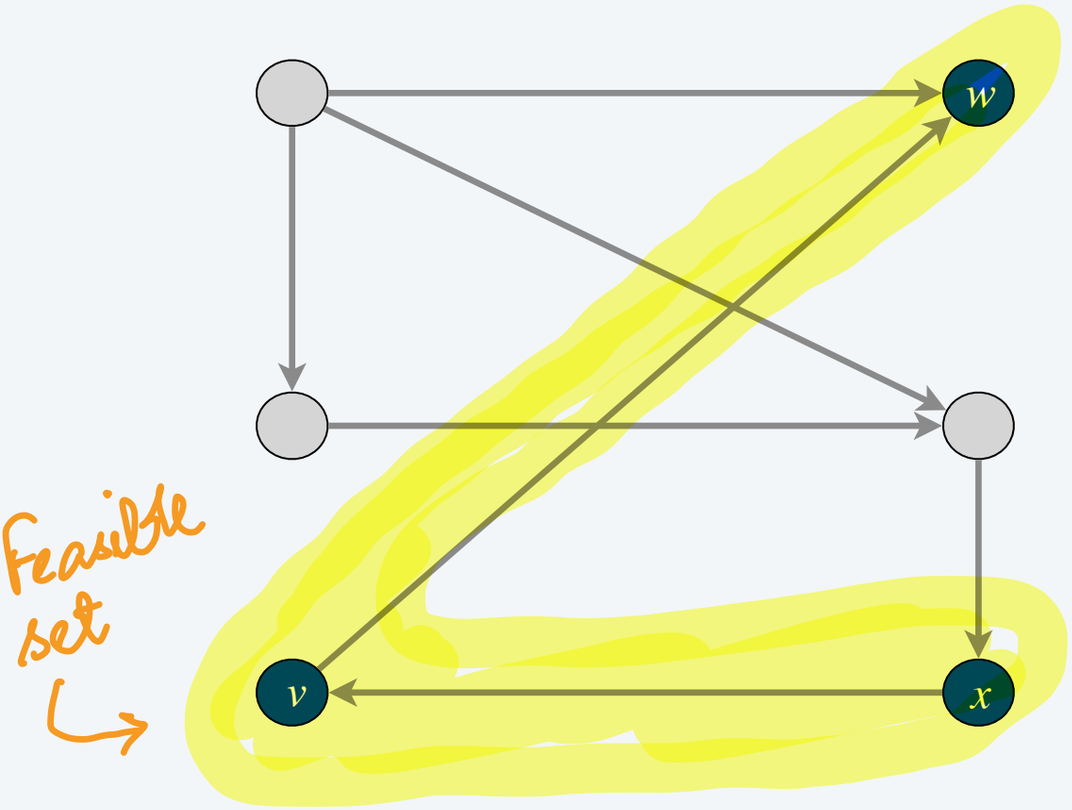
Ecole Polytechnique, Montreal

This paper generalizes the selection problem discussed by J. M. Rhys [12], J. D. Murchland [9], M. L. Balinski [1] and P. Hansen [4]. Given a directed graph G, a closure of G is defined as a subset of nodes such that if a node belongs to the closure all its successors also belong to the set. If a real number is associated to each node of G a maximal closure is defined as a closure of maximal value.

Project selection: prerequisite graph

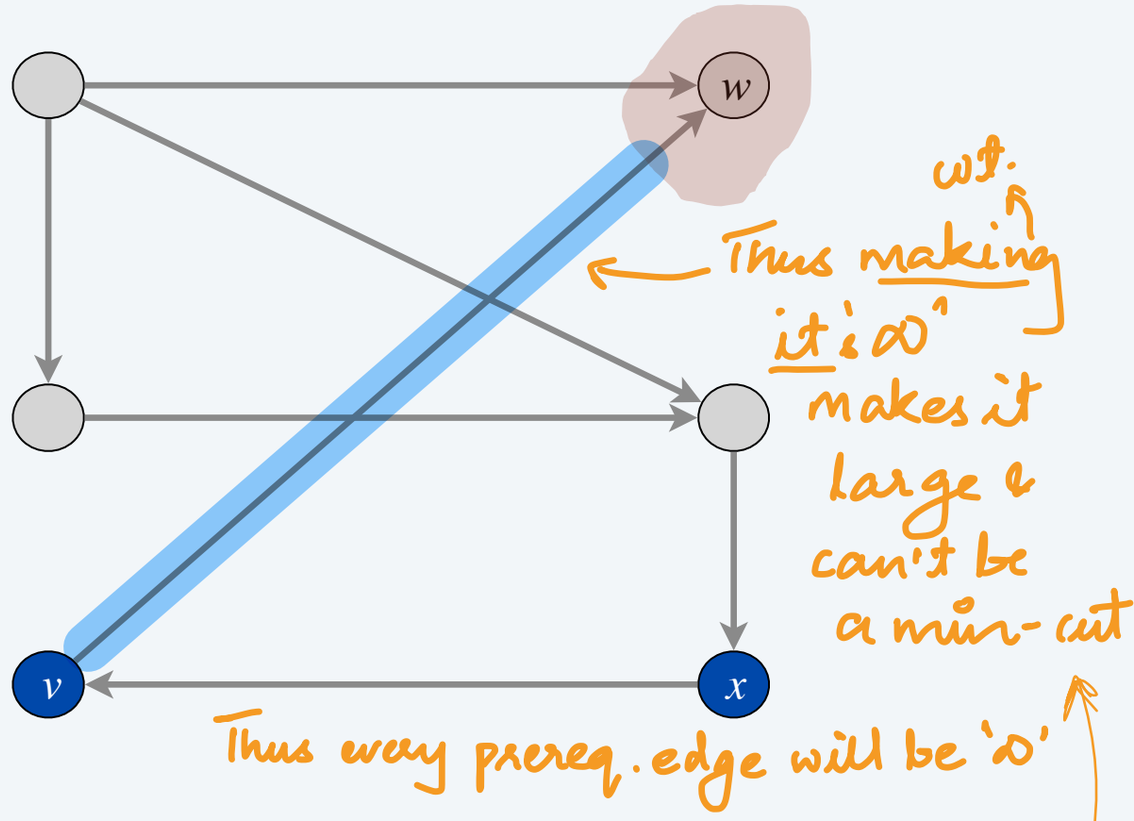
Prerequisite graph. Add edge (v, w) if w is a prerequisite for v.

start from goal and find all prereq.



{ v, w, x } is feasible

V is a prereq. of 'x'
W " " " " 'V'



{ v, x } is infeasible

'How do we define capacity?'
Feasible set will have no outgoing edge!

Project selection: min-cut formulation

Min-cut formulation.

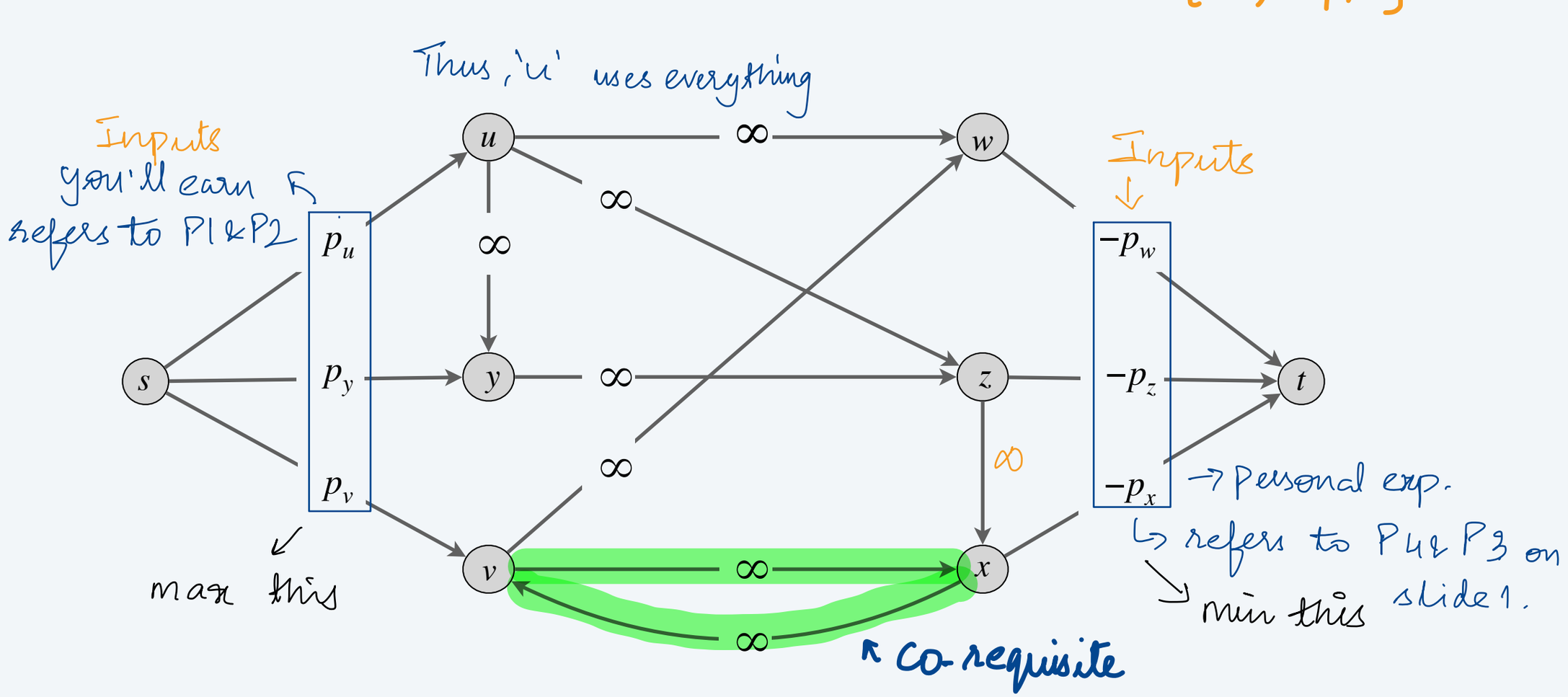
- Assign a capacity of ∞ to each prerequisite edge.
- Add edge (s, v) with capacity p_v if $p_v > 0$.
- Add edge (v, t) with capacity $-p_v$ if $p_v < 0$.
- For notational convenience, define $p_s = p_t = 0$.

↔ any min-cut is a feasible set!

, p_v: how much I'll get after complet^n 'v'.

{u, w, y, z, x, v}

{v, w, x}



Project selection: min-cut formulation

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

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

Min

$$\sum_{v \in V : p_v > 0} p_v - \left[\sum_{v \in A : p_v > 0} p_v + \sum_{v \in B : p_v < 0} (-p_v) \right]$$

$$\sum_{v \in B : p_v > 0} p_v + \sum_{v \in A : p_v < 0} (-p_v)$$

'only outgoing edges!'

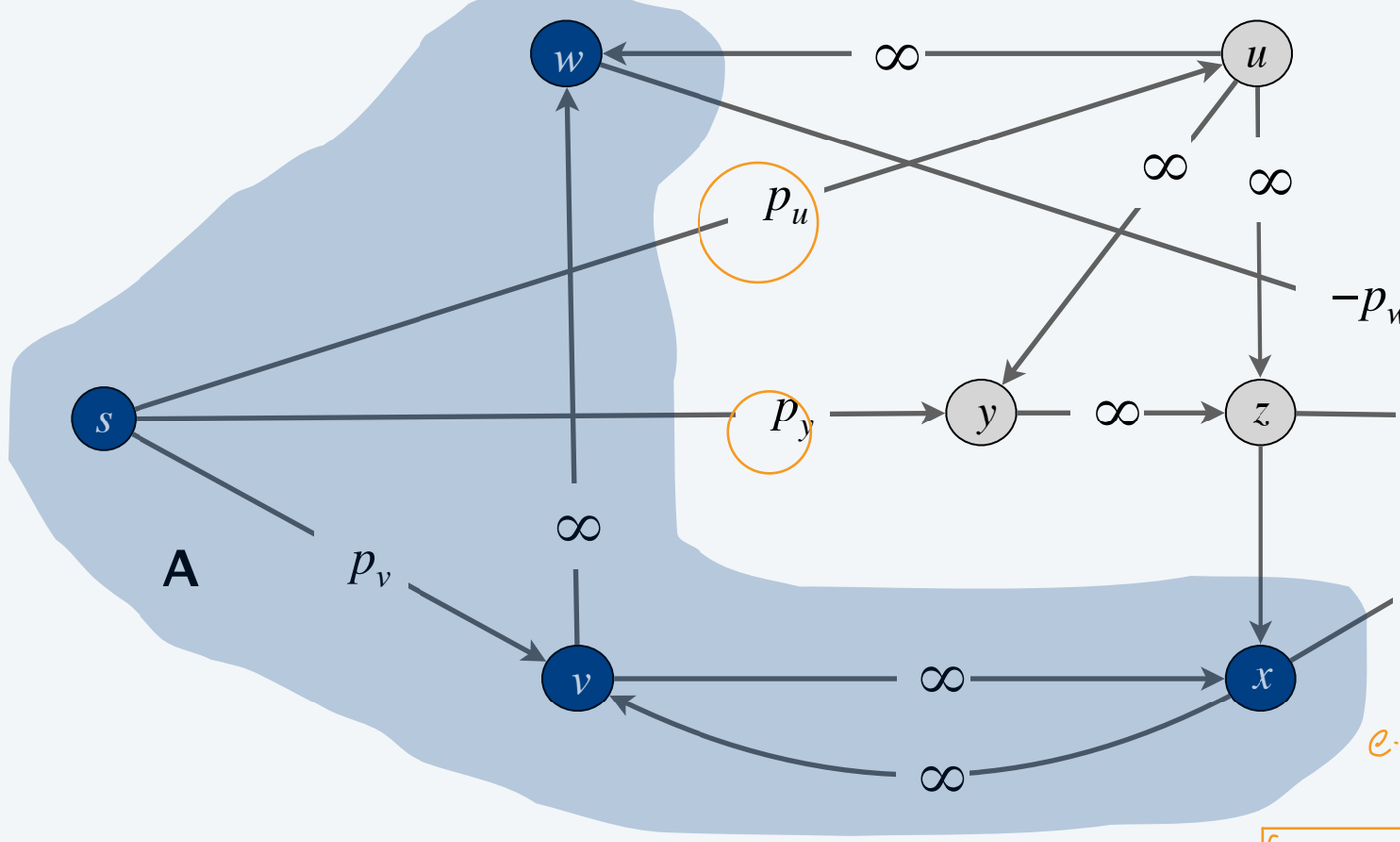
same as image segmentation!

$$cap(A, B) = \sum_{v \in B : \, p_v > 0} p_v + \sum_{v \in A : \, p_v < 0} (-p_v)$$

$$= \sum_{v \, : \, p_v > 0} p_v - \sum_{v \in A} p_v$$

a constant

minimizing this is equivalent to maximizing revenue



Minimize :-
P_u, P_y → missed revenue
P_w, P_x – personal expense.

↓

Maximise revenue

e.g. {P1, P3, u}
$3k + (-500) + (-500) = $2k

{v, w, x} feasible set! → Net rev. :- $P_v + P_w + P_x$

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 = value of ore − processing cost.
- Can't remove block v until both blocks w and x are removed.